



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

UNIVERSITY OF ALBERTA

INTELLIGENT OPERATION SUPPORT SYSTEM FOR
A BATCH SULPHITE PULPING PROCESS

BY

MR. JEAN CORBIN



A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND
RESEARCH IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR
THE DEGREE OF *MASTER OF SCIENCE*

IN

PROCESS CONTROL

DEPARTMENT OF *CHEMICAL ENGINEERING*

EDMONTON, ALBERTA

(SPRING 1992)



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-73250-4

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: *JEAN CORBIN*

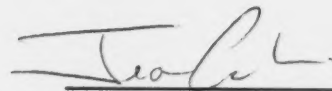
TITLE OF THESES: *INTELLIGENT OPERATION SUPPORT SYSTEM
FOR A BATCH SULPHITE PULPING PROCESS*

DEGREE: *MASTER OF SCIENCE*

YEAR THIS DEGREE GRANTED: *1992*

PERMISSION IS HEREBY GRANTED TO THE UNIVERSITY OF ALBERTA LIBRARY TO REPRODUCE SINGLE COPIES OF THIS THESES AND TO LEND OR SELL SUCH COPIES FOR PRIVATE, SCHOLARLY OR SCIENTIFIC RESEARCH PURPOSE ONLY.

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESES NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.



MR. JEAN CORBIN
29 VIMY STREET
EDMUNDSTON,
NEW BRUNSWICK
CANADA
E3V 1B4

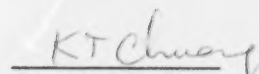
Date: *March 16th 1992*

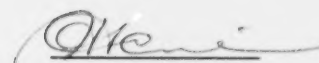
UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

THE UNDERSIGNED CERTIFY THEY HAVE READ, AND RECOMMEND TO
THE FACULTY OF GRADUATE STUDIES AND RESEARCH FOR
ACCEPTANCE, A THESIS ENTITLED **INTELLIGENT OPERATION SUPPORT
SYSTEM FOR A BATCH SULPHITE PULPING PROCESS** SUBMITTED BY
MR JEAN CORBIN IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF **MASTER OF SCIENCE IN PROCESS CONTROL**.


Dr. Ming Rao


Dr. Karl T. Chuang


Dr. Hani Henein

Date: March 16, 92

ABSTRACT

Pulp and paper processes are very complicated, in which the advanced technology and operation experience are highly concentrated. Fundamental requirements of pulp and paper mills are: good product quality, low production costs, high productivity, efficient manufacturing operation, safety of operation and protection of equipment, and utility the most advance technology.

This thesis describes the development of an integrated intelligent system for the operational support of a batch sulphite pulping process. The integrated intelligent system consists of several symbolic reasoning systems and numerical computation packages. Two kinds of software programs, which are developed to predict the final pulp quality by using both qualitative physics (symbolic reasoning systems) and a mathematical model (numerical computation packages), are written in different languages and used independently. All the programs are under the control of a supervising intelligent system, namely, a meta-system. The meta-system manages the selection, operation and communication of these programs.

The system, which is called "Intelligent Operation Support System" (IOSS) for batch pulping processes, provides assistance to human operators to achieve better operation of the batch pulping process. IOSS is the first prototype system developed for the pulping industry. IOSS is constructed using real industrial process data, so that it can be directly implemented in the plant. IOSS, which is a user friendly menu-driven system and a graphic environment, utilizes variable

rules to organize its knowledge base. It also have a common sense reasoning mechanism to protect the knowledge from processing impossible physical information. IOSS, which can be run on any PC/MSDOS base machine, can also be employed for on-line supervisory control.

ACKNOWLEDGEMENT

I would like to express my appreciation towards Dr. Ming Rao, Associate Professor of Chemical Engineering at University of Alberta, who gave me the opportunity to learn, and trains me as a professional researcher. Throughout my continuing training process, I have greatly benefited from his important advice and suggestions, very patient observations and help, strong encouragement and support.

I would also like to thank Dr. K.T. Chuang, Professor of Chemical Engineering, Dr. Hani Henein, Professor of Metallurgical Engineering, and Dr. M.C. Williams, Professor of Chemical Engineering, for their expert advice, important suggestions and patient explanations to help me finish my thesis.

I would like to express my gratitude to Fraser Inc, Edmundston, New Brunswick, Mr. Andrew Slowik, Manager, Process Systems at Fraser Paper Ltd., Madawaska, Maine, USA, and Mr. Gilles Volpe, Technical Engineer (sulphite mill) at Fraser Inc. gave me important suggestions and took part in the development and evaluation of the IOSS.

I sincerely thank Dr. W.K. Nader, Professor of Chemical Engineering, University of Alberta, for helping me to revise parts of my thesis. Also, I would like to thank Professor H. Qiu, Mr Y. Ying, Dr. Q. Xia, Dr. X. Shen, Dr. Q. Wang, and Mr. R.G. Dong of Intelligence Engineering Laboratory at University of Alberta for making important suggestions and comments for the completeness of this project.

My thesis research is financially supported by the Natural Sciences and Engineering Research Council (NSERC), Central Research Fund, Department of Chemical Engineering, The University of Alberta, and Canadian Pulp and Paper Association (CPPA).

TABLE OF CONTENT

	PAGE
CHAPTER 1:	
INTRODUCTION	1
1.1 Motivation	1
1.2 Scope of Study	3
 CHAPTER 2:	
CHEMICAL PULPING	5
2.1 Background	5
2.1.1 Wood chips	6
2.1.2 Chemical pulping operation	8
2.1.3 Kappa number	11
2.1.4 Batch vs continuous process	12
2.1.5 Operating procedure for "batch" digesters	13
2.2 Mathematical models	17
2.2.1 Models for "kraft" pulping	17
2.2.2 models for "sulphite" pulping	23
 CHAPTER 3:	
KNOWLEDGE PROCESSING	28
3.1 Overview of AI	28
3.2 Problem-Solving	30
3.3 Expert System	33
3.3.1 Definition	33
3.3.2 Development procedure for expert system	38
3.3.3 Benefits & advantages	41
3.3.4 AI applications in the pulp and paper industry	42
 CHAPTER 4:	
PROBLEM DEFINITION	43
4.1 Process Operation	43
4.2 Process Analysis	49
4.2.1 Objectives	49
4.2.2 Problems	50
4.3 Problem-Solving	53
4.3.1 Strategy	53
4.3.2 Implementation procedures	54

CHAPTER 5:	
KNOWLEDGE ACQUISITION AND REPRESENTATION	58
5.1 Objectives	58
5.2 Knowledge Acquisition	59
5.3 Qualitative Knowledge Representation	62
5.4 Quantitative Knowledge	78
 CHAPTER 6:	
SYSTEM CONSTRUCTION	81
6.1 Overall System Structure	82
6.1.1 Implementation Platform	82
6.1.2 Programming Languages	82
6.1.3 Function Modules of IOSS	83
6.2 Database	87
6.3 Knowledge Base Organization	89
6.3.1 Overall Structure	89
6.3.2 Qualitative Reasoning System	96
6.3.3 Advanced System Features	98
6.4 Quantitative Computation System Features	104
6.4.1 Computation Process	104
6.4.2 Calculation of "cooking time"	106
6.4.3 Estimation of Kappa Number	107
6.5 Phases of Implementation	108
6.6 Evaluation	109
 CHAPTER 7:	
ILLUSTRATIONS	111
7.1 Qualitative Reasoning	111
7.2 Quantitative Computation	113
 CHAPTER 8:	
CONCLUSION	130
 BIBLIOGRAPHY	134
 APPENDIX	
SYSTEM SOURCE CODE	139

LIST OF TABLES

		PAGE
Table 2.1	Definitions of "kraft pulping" chemistry	9
Table 2.2	Comparison of different cooking process	12
Table 4.1	Performance of the cooking process operation	53
Table 5.1	Desired conditions for pulping quality variables	64
Table 5.2	Description of each situation	69
Table 6.1	The set-up of database array	87
Table 6.2	The set-up of the s_fact array	88
Table 6.3	Description of each situation at level I	100
Table 6.4	Cooking time ranges for the coupled weight values	101
Table 6.5	Number of all possible situations	102

LIST OF FIGURES

	PAGE
Figure 2.1 Materials flow chart of a chemical pulp & paper mill	7
Figure 2.2 Wood chip	11
Figure 2.3 Digester flow diagram	15
Figure 2.4 Temperature profile of a batch cycle operation	16
Figure 2.5 Relative reaction rate vs cooking time	18
Figure 3.1 The three components in "Problem-Solving"	32
Figure 3.2 Relationship between "AI" and "Expert System"	32
Figure 3.3 Architecture of "Production System"	34
Figure 3.4 Standard stages for the development of "Expert System"	39
Figure 4.1 "Temperature & Pressure" curves of a cooking cycle	48
Figure 4.2 Frequency of "Kappa Numbers"	52
Figure 4.3 Information flow of an overall system	57
Figure 5.1 Flow of knowledge for the development of the IOSS	60
Figure 5.2 Qualitative knowledge for "Chip Load"	70
Figure 5.3 Qualitative knowledge for "Chip Quality"	71
Figure 5.4 Qualitative knowledge for "Chip Size"	72
Figure 5.5 Qualitative knowledge for "Total SO ₂ "	73
Figure 5.6 Qualitative knowledge for "Free & Combined SO ₂ "	74
Figure 5.7 Qualitative knowledge for "pH" of cooking acid	75
Figure 5.8 Qualitative knowledge for "Operating Pressure"	76
Figure 5.9 Qualitative knowledge for "Operating Temperature"	77

Figure 6.1	Reasoning process of "qualitative system"	97
Figure 6.2	Classification of "knowledge base	99
Figure 6.3	Computation process of "quantitative system"	105
Figure 7.1	Input data for "total SO ₂	115
Figure 7.2	Display of qualitative reasoning results for "chip load"	116
Figure 7.3	Display of qualitative reasoning results for "total SO ₂ "	117
Figure 7.4	Display of qualitative reasoning results for "operating. temp."	118
Figure 7.5	Display of qualitative reasoning results for "operating press."	119
Figure 7.6	Selection for graphical display of "cooking times"	120
Figure 7.7	Estimated "cooking time for each individual inputs variables	121
Figure 7.8	Estimated "cooking time based on selected coupled inputs variables	122
Figure 7.9	Estimated "cooking time" based on all input variables coupled together	123
Figure 7.10	Selection to access to quantitative computation system	124
Figure 7.11	Input data for quantitative computation system	125
Figure 7.12	Display of the required target "S _m -factor"	126
Figure 7.13	Display of the estimated "cooking time"	127
Figure 7.14	Display of the estimated "remaining cooking time" and "kappa number"	128
Figure 7.15	Graphical representation of the calculated total "cooking time"	129
Figure 8.1	Types of users for IOSS	133

LIST OF SYMBOLS

A	=	Parameter, [B/373] (equation 2.18)
A	=	Initial lignin content of wood chips (equation 2.10)
Ac	=	Active alkali charge
B	=	Activation energy parameter(function of wood species) (equation 2.18)
B	=	Conversion rate factor (equations 2.10 & 2.13)
C	=	Concentration of cooking liquor (equation 2.19)
C	=	Acid concentration (equation 2.20)
C	=	Active chemical concentration of cooking liquor (equations 2.4 to 2.8)
Co	=	Initial concentration of active chemical in cooking liquor
D	=	Percent total SO ₂ of acid (equation 2.20)
D	=	Initial concentration of active chemicals of cooking liquor (equations 2.6 & 2.7)
E	=	Activation energy
F	=	Frequency factor, constant
H	=	H-factor
K	=	Kappa number of pulp
K's	=	Constant (equation 2.3)
L	=	Lignin content of pulp
L ₀	=	Initial lignin content of wood chips
PSO ₂	=	Partial pressure for SO ₂ inside the digester
P _{tot}	=	Total pressure in the digester
R	=	Universal gas constant (equation 2.2)
R	=	Black liquor residual alkali (equations 2.11 & 2.12)
S	=	S-factor
T	=	Absolute cooking temperature

- a_x = Regression coefficients
- k = Relative reaction rate
- $k(T)$ = Temperature-dependent rate constant for lignin dissolution
(equations 2.14 & 2.15)
- t = Time during cooking cycle
- t_1 = The time when the cooking liquor sample is taken for total SO_2 (10)
- r_L = Rate of lignin dissolution

Greek symbols

- α = Parameter, constant
- β = Parameter, constant
- γ = Parameter, constant
- λ = Parameter, constant
- σ = Parameter, constant
- τ = First order reaction constant characteristics

Abbreviation

- AI = Artificial intelligence
- ES = Expert System
- IOSS = Intelligence Operation Support System
- S_m -factor = Modified S-factor
- S_{mt} -factor = Target modified S-factor
- S_{m1} -factor = Modified S-factor at the time the cooking liquor sample is taken for total % SO_2 (10)
- S_{m2} -factor = Modified S-factor during constant temperature and constant pressure

CHAPTER 1

INTRODUCTION

Artificial Intelligence (AI) is a growing field that is making slow but steady progress in the chemical engineering area [Ferrada, 1990]. It is applied to chemical process industries to simulate human skills and knowledges for a variety of areas, such as process operation, process control, training, maintenance, etc. In the pulp and paper industry, recent modernization and automation of the industrial base aimed at increasing productivity and profitability have imposed much greater demands on all level of human skills [Waye, 1987].

In this chapter, the importance of AI technique for chemical processes is presented first. The scope of this project and organization of the thesis are then described.

1.1 MOTIVATION

The main objective of most chemical processes is to convert raw materials into high quality products at the lowest cost. Another important issue concerning chemical processes is their impact on the environment. Our society and governments do not tolerate the construction and/or the operation of a manufacturing plant that is harmful to the environment. However, a chemical

plant that is well designed, well controlled, and well operated, should meet the above requirements.

The safe operation of a chemical process depends on the human operator and the real-time process control system; both of them must perform well together [Waye, 1991]. For processes which are well described by mathematical models and have very well designed real-time control systems; little human skill and expertise are usually required for the desired operations. However, many chemical industrial processes are very complex in nature, the development of mathematical models for process control is difficult, and could result in a poor performance of the real-time control system. In such a case, only the most experienced and skilled operators can handle the operation of the process.

It is clear that the more complex a process is, the more human expertise and human skill are required for rigorous process operation and monitoring. This means that the quality of a product produced from a chemical process becomes strongly dependable on the skill of human operators.

Since today's computers are extremely powerful at affordable prices, Artificial Intelligence (AI) techniques are being used more and more by researchers and engineers. It is suggested that AI techniques could be used to develop intelligent systems for the operation and control of chemical processes at the human operation level. As a sub-domain of AI research and development, expert systems provide a programming methodology for solving ill-structured problems which are too difficult to be handled by purely numerical algorithm methods. It is one way to assist human operators in evaluating the data and to advise them how to react to changing conditions. Since expert systems

incorporate the knowledge of experienced operators and designers, the operators have the benefit of expert advice on short notice at all times. The development of expert systems in the real world requires understanding of both computer science, and even more important domain knowledge, for example, chemical engineering [Borsje, 1990].

1.2 SCOPE OF STUDY

One of the most important industries in Canada is the pulp and paper industry. High quality pulp is often produced by a chemical process in a batch operation. This chemical process utilizes cooking liquor or cooking acid to dissolve and remove the lignin from wood chips for the production of chemical pulp. Its primary objective is to cook wood chips to a desirable degree of delignification (i.e. desired *kappa number*). When a product is produced in a batch operation, the desired quality of the product is only reached at the end of the batch cycle.

As batch digesters operate at high temperature (e.g. 150-170 °C) and high pressure (e.g. 600-700 kPa), it is not practical to remove a pulp sample and analyze its quality (*kappa number*) on-line. For these types of processes, the human operator becomes a very important factor for monitoring/control batch digesters. Operators need to use their experience and skills to estimate the pulp quality or rather to estimate the time to terminate the cooking process in order to produce good quality pulp.

This thesis describes the construction of an *Intelligent Operation Support System (IOSS)* for a batch chemical pulping process. Such a system contains a qualitative reasoning system and a quantitative computing system. The system is to be used by the human operators to obtain suggestion, conclusion, and

recommendation for the production of good quality pulp. IOSS can provide estimated "cooking time", estimated kappa numbers, and other valuable information such as faults diagnosis to the operators. The main objective of IOSS is to help operators in decision-making. However, IOSS may also be used for training engineers and new operators.

The thesis is organized in the following manner. Chapter 2 presents the background about chemical pulping. Chapter 3 provides an overview of knowledge processing techniques. In Chapter 4, a real industrial pulping process for the application of an IOSS is described. Chapter 5 presents the required qualitative knowledge and the quantitative knowledge acquired from experience and from the experts (operators) for the construction of IOSS. Chapter 6 describes the construction of IOSS for the process discussed in chapter 4. Chapter 7 is an illustration of IOSS followed by a conclusion in chapter 8.

CHAPTER 2

CHEMICAL PULPING

This chapter presents the background theory describing the main facets of the chemical pulping processes. It is important for readers to understand the chemical pulping process before discussing the thesis research project. Therefore, a brief discussion concerning the chemical components of wood chips and different types of chemical processes will be presented. In addition, a description of the operation of the cooking vessel (digester) and a literature survey section completes this explanation.

2.1 BACKGROUND

Basically, there are three main types of pulping processes. These processes are chemical, mechanical, and a combination of both. In the chemical pulping, the chips are separated into fibers by digesting (cooking) in a pressurized vessel with chemicals at high temperatures. Mechanical pulp is produced either by grinding the logs into separated fibers with rotating stones, and/or by forcing chips between grooved rotating steel plates in refiners. In the third process the chips are first treated with steam and chemicals before going through the refiners. This type of process is the most recent one and is

sometime called "BCTMP" which stands for Bleach-Chemical-Thermo-Mechanical-Pulp.

Among the three processes, the purely chemical process produces a product with highest pulp quality, but with lowest yield (50-60%). Chemical pulp is mainly used to produce high quality paper for magazines and books. The mechanical process, which produce the lowest pulp quality product, transforms over 90% of the debarked wood into useful pulp. This type of pulp is generally used for newsprint.

The chemical pulping process is the one that was selected. Figure 2.1 shows a general flowchart of the process from roundwood to paper. The digester, which is shown with a dotted rectangle, is the focus of this thesis.

The following sub-sections will discuss some useful fundamental concepts involved in the production of chemical pulp.

2.1.1 Wood Chips

Wood can be split into two main categories; softwood (e.g. spruce, fir, pine, etc) and hardwood (e.g. maple, oak, etc). It is basically constituted of three types of components which are polysaccharides (cellulose 40-50 wt% & hemicellulose 0-30 wt%), lignin (15-35 wt%) and extractive substances (3-10 wt%) [Smook, 1982]. Cellulose $(C_6H_{10}O_5)_n$ is a polymer of glucose that determines the character of the fiber and permits its use in papermaking. Hemicellulose is another type of polymer. It is made of five different kinds of sugars which are glucose, mannose, pentose, galactose and arabinose. These sugars are combined with uronic acids to form different polymeric structures.

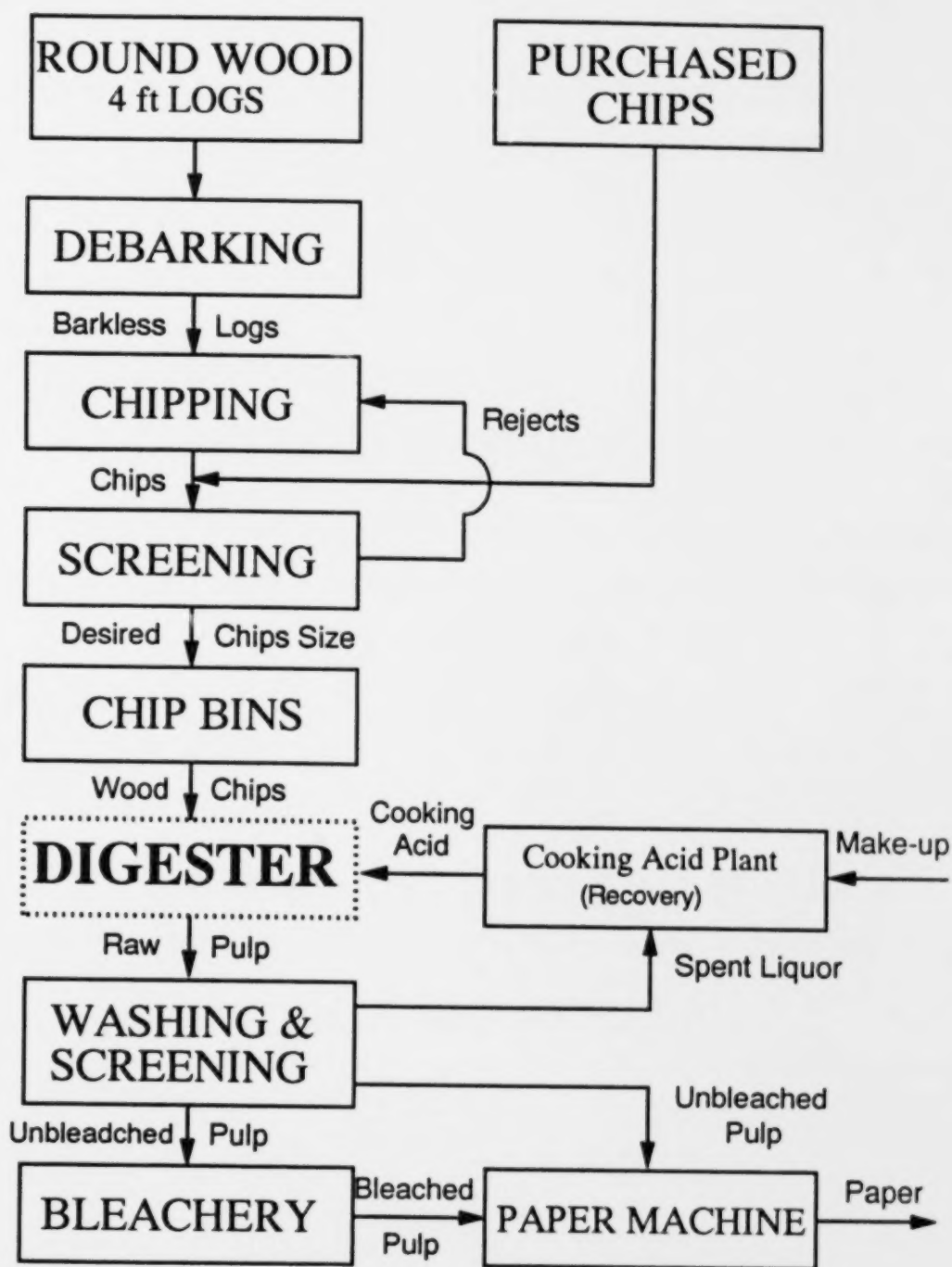


FIGURE 2.1 Materials flow chart of a chemical pulp & paper mill

Lignin is a complex chemical substance which is highly polymerized. It forms the outer layer of the fibers called the "middle lamella". Basically, lignin is concentrated in the fibers cells and cements the fibers together, giving the tree structural rigidity (see figure 2.2).

Extractives are other different substances in wood chips. Depending on the plant species, these substances are resin acids, fatty acids, turpenoid compounds and sterols [Smook, 1982].

2.1.2 Chemical Pulping Operation

In the chemical pulping processes, individual wood fibers are loosened and separated from each other by dissolving and removing the lignin which holds them together. Cooking liquor is the acid or the base that is utilized to remove the lignin from the fibers by dissolving it under high pressure and high temperature.

There are two different types of chemical processes for the production of chemical pulp. They are "*Kraft pulping*" and "*Sulphite pulping*".

In Kraft pulping, the active cooking liquor used for delignification is an alkaline solution. This solution, which is frequently called "white liquor", contains sodium hydroxide (NaOH) and sodium sulfide (Na_2S) as active agents. The residual liquor, often called black liquor, contains the reaction products. These products are burned in a recovery unit to produce an inorganic smelt sodium carbonate (Na_2CO_3) and sodium sulfide (Na_2S). To recover the white liquor, the smelt is dissolved to form the "green liquor" which is reacted with CaO to convert the Na_2CO_3 into NaOH . The following table provides the standard definition for chemical Kraft pulping

Table 2.1 Standard definition of kraft pulping chemistry [McCubbin, 1983]

Active Alkali	$\text{NaOH} + \text{Na}_2\text{S}$
Effective Alkali	$\text{NaOH} + 1/2 \text{Na}_2\text{S}$
Sulphidity	$\frac{\text{Na}_2\text{S}}{\text{NaOH} + \text{Na}_2\text{S}} \times 100$
Total Alkali	$\text{NaOH} + \text{Na}_2\text{S} + \text{Na}_2\text{CO}_3 + \text{Na}_2\text{SO}_4$
Total Titratable Alkali	$\text{NaOH} + \text{Na}_2\text{CO}_3 + \text{Na}_2\text{S}$
Causticity	$\frac{\text{NaOH}}{\text{NaOH} + \text{Na}_2\text{CO}_3} \times 100$
Activity	$\frac{\text{NaOH} + \text{Na}_2\text{S}}{\text{NaOH} + \text{Na}_2\text{S} + \text{Na}_2\text{CO}_3} \times 100$

The main disadvantage of this process is that a relatively large amount of residual condensed lignin remains in the pulp and cannot be removed by further cooking without the degradation of the fibers (carbohydrates). On the other hand, because this process utilizes an alkaline solution (high pH), extractives of certain wood species which cannot be removed in acid sulfite pulping are dissolved and removed from wood by the alkaline cooking liquor. The unremoved extractives will cause the problems with pitch on paper machines located downstream.

In sulfite pulping, the cooking liquor is prepared by burning sulfur to produce SO_2 gas. The SO_2 gas is then absorbed in an alkaline base solution for the production of the raw cooking acid. The most commonly used bases are NH_4OH , Na_2CO_3 , or $\text{Mg}(\text{OH})_2$. The raw cooking liquor contains a mixture of free SO_2 and combined SO_2 in a specific proportion for a total concentration of 5% to 9%. The combined SO_2 , which is usually at a 2% to 3% concentration, is in the form of bisulphite. Depending on the base used, the bisulphite is in the form of

ammonium bisulphite [NH_4HSO_3], sodium bisulphite [NaHSO_3], or Magnesium bisulphite [$\text{Mg}(\text{HSO}_3)$]. The pH of the liquor is controlled by adding the base. This cooking liquor is mixed with wood chips in a reactor vessel (digester) for delignification. High pressure and high temperature are both required in order to proceed reaction and to dissolve the lignin. In sulphite processes, the main reaction takes place between the bisulphite ions and the lignin in the wood.

For chemical pulping processes, the chemical reactions involved in delignification of wood chips are very complex and not yet completely understood [Balchen et al, 1988]. However, the reaction between wood chips and cooking liquor at high pressure and high temperature can be described in the following manner:

- i) transport of the ions in the liquor into the chips
by penetration and diffusion,
- ii) adsorption of the ions onto the surface of the
wood constituents,
- iii) reaction between the wood constituents and
the adsorbed ions (selective),
- iv) desorption of the reaction products,
- v) transport of the reaction products to the
exterior of the chips by diffusion.

2.1.3 Kappa Number

The kappa number test is a measurement of the lignin content of the pulp and is used by mill operators to control and determine the degree of delignification. An acid permanganate solution (KMnO_4) of known volume is used to react with lignin in a pulp sample. The amount of KMnO_4 consumed is determined by titration, and the kappa number is calculated [West et al., 1987]. Usually, the desired kappa number is specified by the mill. The operator controls the cooking process in order to achieve the specified target kappa number (i.e. a certain degree of delignification). Depending upon the mill and the process, a target kappa number usually ranges between 25 and 30.

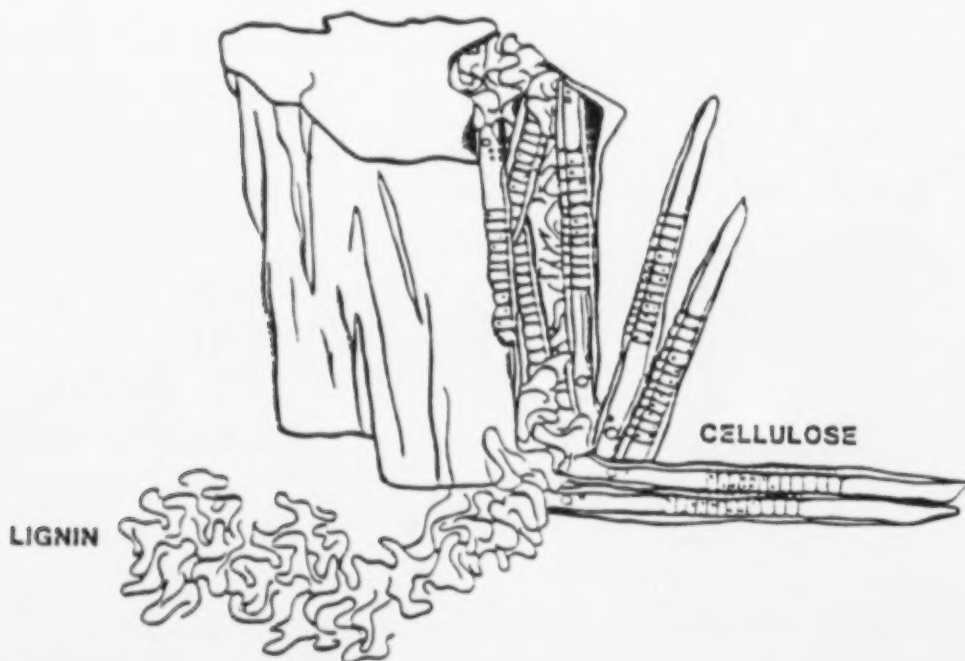


FIGURE 2.2 Wood chip

2.1.4 Batch vs Continuous Processes

Most of the old chemical pulp mills use batch digesters. The newer mills use continuous ones. Both of them produce the same type of pulp. In table 2.2, a survey [Patton, 1977] shows advantages and disadvantages for both batch and continuous processes.

Table 2.2 Comparison of Different Cooking Processes:

<u>PROCESSES</u>	<u>Factors Favoring the Corresponding Process</u>
Continuous Digesters	1. Less Energy Requirements 2. Less Pollution Discharges
Batch Digesters	1. Flexibility 2. Production Reliability 3. By Product Yield 4. Lower Maintenance Requirements
Both Continuous & Batch Digesters	1. Pulp Quality & Uniformity 2. Chemical Requirements 3. Personnel Requirements 4. Capital Cost

It can be seen from table 2.2 that the batch digesters are in favor for 8 out of 10 factors. Because the continuous digesters have more mechanical equipment; maintenance and repair requirements are usually higher than the batch digesters [Patton, 1977]

One pulping operation line is normally designed with one continuous digester or with several batch digesters. If the continuous digester is out of control, or down, the complete line is down, whereas if one batch digester is down, other digesters are still working to keep the operation running. This is why batch digesters are better and more practical for production reliability. Also, when frequent changes of wood species are required, batch digester are more flexible than continuous digesters.

The attractive features of continuous digesters are lower impact on the environment, less energy requirement and the utilization of more advanced technology.

2.1.5 Operating Procedure for a Batch Digester

Figure 2.3 shows a typical batch digester that is a large pressure reactor vessel equipped with pumps and valves for handling the wood chips, cooking liquor, steam and pulp. For indirect system, the cooking liquor is circulated through a heat exchanger (liquor heater) to raise and maintain the temperature. Usually there are several identical batch digesters in a pulp mill. This makes it possible to select the cooking cycles, usually about 6-8 hours, to maintain a steady flow of pulp from the pulping process. A typical operation cycle follows five general steps. These steps are listed as follows:

- | | |
|--------|---|
| Step 1 | Filling the digester with wood chips |
| Step 2 | Filling the digester with cooking liquor |
| Step 3 | Bringing the digester and contents up to operating temperature by the addition of steam |

- Step 4 Cooking for a certain time under desired pressure and temperature
- Step 5 Dumping/Blowing the digester (removing the pulp from the digester)

Usually, different mills have different operating conditions and process characteristics. Figure 2.4 shows a cooking temperature curve of a process operating at 170 °C, a rising time (heating time) of 90 minutes plus 90 minutes of cooking at operating temperature, for a total cooking time of 3.0 hours. When the target pulp quality is reached, the digester is emptied (step 5).

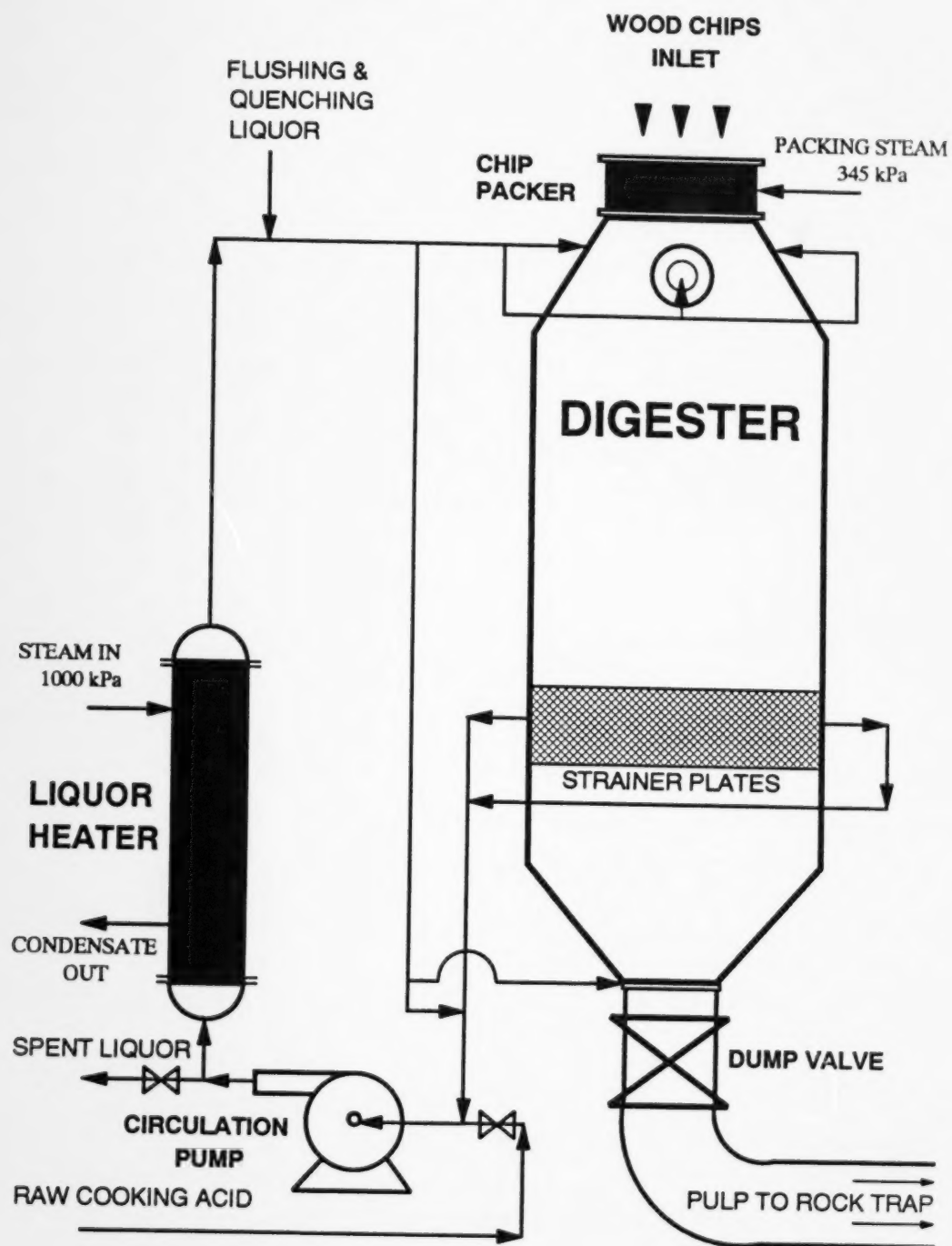


FIGURE 2.3 Digester flow diagram

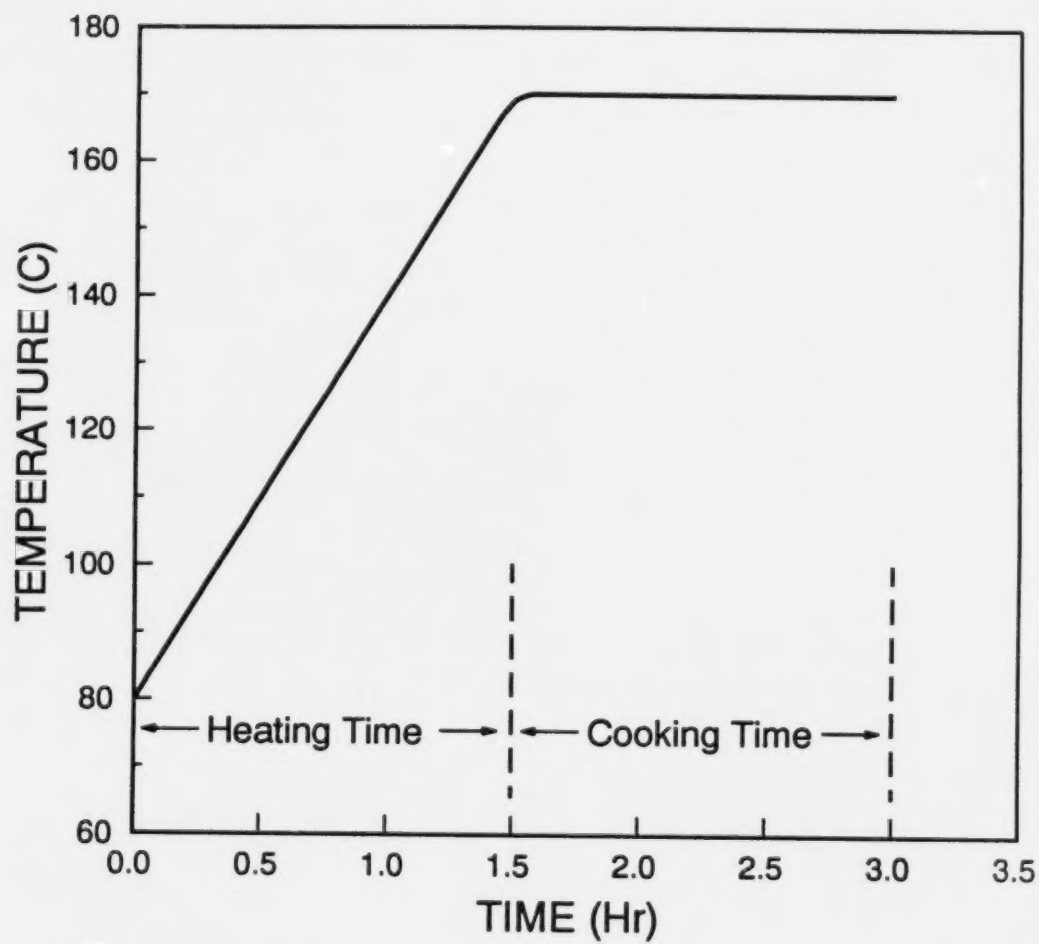


FIGURE 2.4 Temperature profile of a batch cycle operation

2.2 MATHEMATICAL MODELS

This section presents pulping mathematical models for batch digester control that have been widely used in both kraft and sulphite processes.

2.2.1 Models for Kraft Pulping

One of the very first paper published on batch digester control was in 1957 by K.E. Vroom. Vroom developed a method to express time and temperature of kraft pulping as a single numerical value for batch digesters. This value is called "*H-factor*", and can be used to predict or estimate the degree of delignification during the cooking cycle. The H-factor represents the area under the curve in which relative reaction rates are plotted against time. The H-factor is shown in Figure 2.5 and is described by the following equation:

$$H = \int_0^t k \, dt \quad (2.1)$$

where

H = H-factor

k = relative reaction rate

t = time of cooking cycle

The relative reaction rate, k, is a temperature dependent term and can be expressed by the Arrhenius equation in the following form:

$$k = e^{\left[F - \frac{E}{R \cdot T}\right]} \quad (2.2)$$

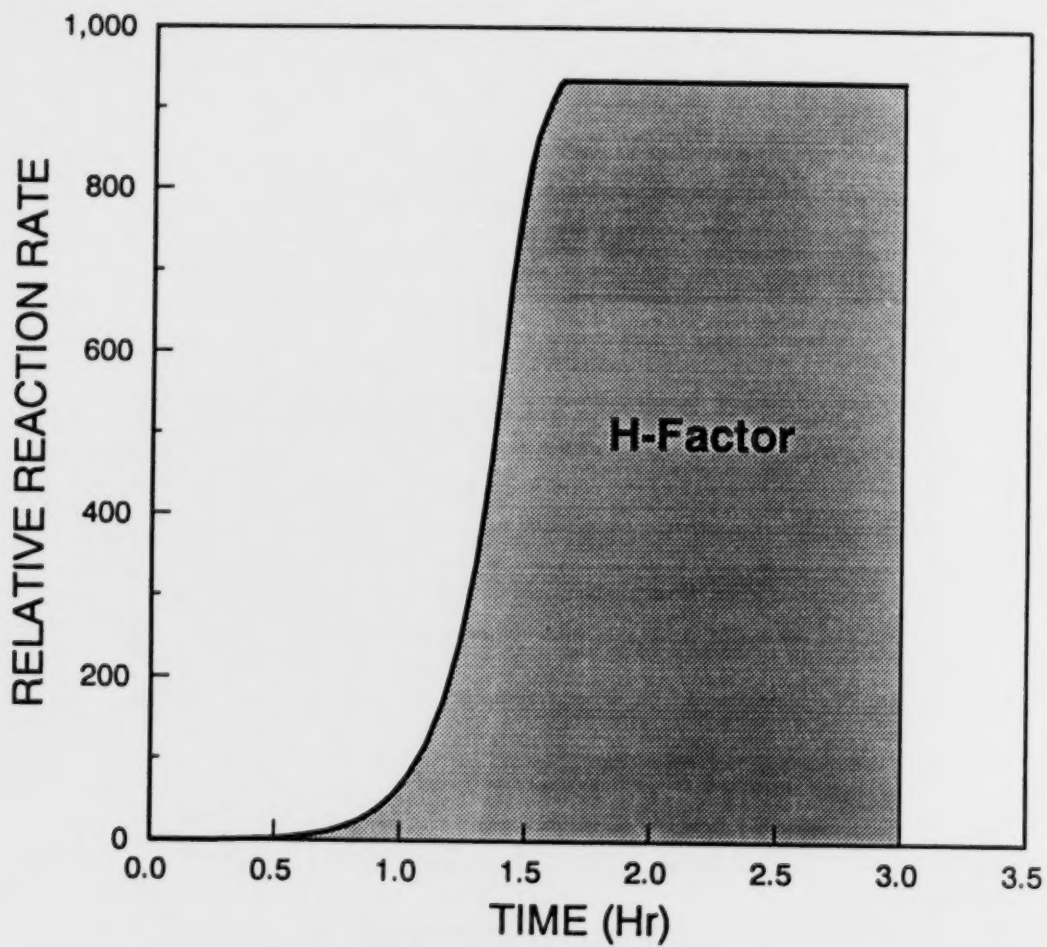


FIGURE 2.5 Relative reaction rate vs time

where

F = frequency factor, constant

E = activation energy

R = gas constant

T = absolute cooking temperature

Usually, the activation energy, E, is determined experimentally. Vroom reported that for kraft pulping of spruce, the activation energy should be approximately 32000 cal/mol to remove the bulk lignin. However, if a different wood species is used, a different value of the activation energy should be employed to calculate the relative reaction rate. A method developed by Wang [1987] based on the temperature profile of the cooking cycle and residual lignin is suggested to determine the numerical value of the activation energy.

According to Vroom [1957], the relative reaction rate of delignification is unity ($k=1$) at 100 °C or 373 K. Therefore, at $T = 100$ °C, using the appropriate universal gas constant ($R = 1.986$ cal/mol-K), the value of the frequency factor F can be obtained by substituting the above numerical values in equation (2.2).

Therefore

$$1 = e^{[F - \frac{32\,000 \text{ cal/mol}}{(1.986 \text{ cal/mol} \cdot \text{K})(373 \text{ K})}]}$$

$$F = 43.20$$

Based on the H-factor, many mathematical models have been developed to adapt different mills or pilot plant digesters [Kleppe, 1970; Freedman, 1976; Kerr, 1976; Beller et al., 1988].

Kleppe reported that an empirical model of the following form [equation (2.3)] was developed for the prediction of the kappa number for a Finnish mill [Kleppe, 1970].

$$\text{Kappa No.} = K_0 - K_1 \cdot \text{Ac} + K_2 \cdot \text{Ac}^2 - K_3 \cdot \text{H} + K_4 \cdot \text{Ac} \cdot \text{H} \quad (2.3)$$

where

H = Vroom H-factor
Ac = Active alkali charge
K's = Constant

It was possible to obtain a regression correlation of 0.98 with their observed kappa numbers and by using the above empirical regression equation.

One of the most recent mathematical model for kraft pulping was developed by Beller et al, [1988]. This equation was based on the assumption that the rate of delignification is a function of the free cooking liquor concentration. Therefore, its signification is that the pulping reaction rate is zero when all the active pulping chemicals are consumed. That is:

$$\frac{dL}{dt} = \lambda \frac{dC}{dt} \quad (2.4)$$

where

C = active chemical concentration of cooking liquor
L = lignin content of pulp
t = time
 λ = constant

It is also assumed that the rate of active chemical consumption of cooking liquor is proportional to the relative reaction rate and to the active chemical in the cooking liquor. This leads to the following relationship:

$$\frac{dC}{dt} = -\alpha \cdot k \cdot C \quad (2.5)$$

where α is a constant

Through manipulation and integration of equation (2.5), a new equation is obtained:

$$C \cdot \exp\left(\alpha \int_0^t k \, dt\right) = D \quad (2.6)$$

Substituting for the H-factor, equation (2.6) can be rewritten as follows:

$$C = D \cdot e^{(-\alpha \cdot H)} \quad (2.7)$$

where $D =$ initial concentration of active chemicals of cooking liquor
 $= C_0$

Integrating equation (2.4) generates the following equation:

$$(L - L_0) = \lambda \cdot (C - C_0) \quad (2.8)$$

where $L_0 =$ initial lignin content of wood chips

Combining equation (2.7) and (2.8) together leads to the following expression:

$$L = L_0 - \lambda \cdot C_0 [1 - e^{(-\alpha \cdot H)}] \quad (2.9)$$

Equation (2.9) is an expression to predict or estimate the lignin content in wood chips during the cooking cycle. For mill purposes, this expression can be modified for the kappa number estimation. Therefore, the process model for the prediction of black liquor residual alkali and the pulp kappa number can be expressed by the following two equations:

$$K = A + B \cdot \{C_0 \cdot [1 - e^{(-H/\tau)}]\} \quad (2.10)$$

$$R = C_0 \cdot [e^{(-H/\tau)}] \quad (2.11)$$

where

- A = Initial lignin content of wood chips
- B = Conversion rate factor
- C₀ = Initial concentration of active chemical in cooking liquor
- H = Vroom H-factor
- K = Kappa number of pulp
- R = Black liquor residual alkali
- τ = 1st order reaction constant characteristics

In order for the above model to adapt the real process, parameters A, B, and τ have to be determined experimentally. If the wood species used for pulping are known, the lignin content of the raw material, A, can be obtain from literature.

Equation (2.12) is an expression derived from equation (2.11) to calculate τ, where

$$\tau = \frac{H}{\ln(C_0 / R)} \quad (2.12)$$

By measuring H and R at several different times during the cooking cycle, numerical values of τ can be obtained for a known value of C_0 . An average τ can then be calculated and implemented into the process model.

Beller demonstrates that the method to be employed to estimate the numerical value of parameter B , consists of using two different cooking conditions for the same wood species. This means that for the same value of A , different values of C_0 and H are utilized for each cooking condition. Using equation (2.10), we can get:

$$B = \frac{K_1 - K_2}{C_{a,1} [1 - \exp(-H_1 / \tau)] - C_{a,2} [1 - \exp(-H_2 / \tau)]} \quad (2.13)$$

where subscripts 1 and 2 represent for cooking condition 1 and cooking condition 2. It should be mentioned that the parameters A , B , and τ are specific for given wood species.

Beller states that the implementation of this model has improved the pulp quality of a real industrial mill. However, much of the success is primarily due to the associated work of the pulp mill modernization and automation.

2.2.2 Models for Sulphite Pulping

The H-factor developed by Vroom for kraft pulping results in the development of many other cooking models. A similar method of calculating the degree of delignification exists for sulfite pulping process. This particular model is the S-factor method [Bylund et al., 1982]. The S-factor includes both the temperature and the partial pressure of SO_2 in the model. Its derivation procedure is given as follow.

The reaction rate of lignin dissolution in sulfite pulping can be written as follows [Bylund et al., 1982]:

$$r_L = \frac{d[L]}{dt} = -k(T) \cdot [\text{HSO}_3^-]^\sigma \cdot [\text{H}^+]^\beta \cdot [L]^\gamma \quad (2.14)$$

where

r_L	=	Rate of lignin dissolution
$[L]$	=	Lignin content in wood residue
$k(T)$	=	Temperature-dependent rate constant for lignin dissolution
$[\text{HSO}_3^-]$	=	Hydrogen sulfite ion-concentration of the cooking liquor
$[\text{H}^+]$	=	Hydrogen sulfite ion-concentration of the cooking liquor
β, σ, γ	=	Parameters

By making the following assumptions, equation (2.14) can be simplified.

1. The reaction orders with respect to $[\text{H}^+]$ and $[\text{HSO}_3^-]$ are of the same magnitude (i.e. $\sigma = \beta$)
2. The product $[\text{H}^+] \times [\text{HSO}_3^-]$ is directly proportional to the partial pressure of SO_2 gas in the digester

According to Bylund [1982], both assumptions are approximately valid. After rearrangement of equation (2.14), the following equation is obtained:

$$\frac{d[L]}{[L]^\gamma} = -k(T) \cdot P^{\text{SO}_2} \cdot dt \quad (2.15)$$

The right hand side of equation (2.15) includes the three main cooking parameters.

T = Temperature of digester content which an Arrhenius expression goes into the reaction rate constant just like in the H-factor for kraft pulping

PSO₂ = Partial pressure for SO₂ inside the digester

t = Time

By making one more assumption, equation (2.15) can be easily integrated.

3. The reaction order with respect to lignin concentration in the wood residue is of first order (i.e. $\gamma = 1.0$)

Then, a new equation is obtained:

$$\ln[L]_t - \ln[L]_0 = -\int_0^t k(T) \cdot P^\sigma \text{SO}_2 \cdot dt \quad (2.16)$$

The S-factor is defined by the following equation:

$$S = \int_0^t k(T) \cdot P^\sigma \text{SO}_2 \cdot dt \quad (2.17)$$

and

$$k = e^{[A - B/T]} \quad (2.18)$$

where

S = S-factor
A = Parameter, (B/373)
B = Activation energy parameter
(function of wood species)

Based on laboratory experimental study, the following numerical values were obtained [Yorston et al., 1965],

$\sigma = 0.75$ to 0.85 bar

B = 10166 Kelvin

Based on experimental trials in sulfite mill at Domsjö of Sweden, Bylund [1982] obtained the following values:

$$\sigma = 0.75 \text{ bar}$$

$$B = 11500 \text{ Kelvin}$$

Empirical models for the prediction of the kappa number has been built using the S-factor for sulfite pulping [Bylund et al, 1982; Haywood, 1989]. These models are shown as follows:

$$\text{Kappa No.} = a_1 + a_2 \cdot C + a_3 \cdot e^{-a_0 S} + a_4 \cdot C \cdot e^{-a_0 S} \quad (2.19)$$

where

S = S-factor

C = Concentration of cooking liquor

a_0, a_1, a_2, a_3, a_4 = Regression coefficient

and

$$\text{Kappa No.} = a_1 - a_2 \cdot P_{\text{tot}} \cdot \int_0^t k \, dt + a_3 \cdot C + a_4 \cdot D \quad (2.20)$$

where

P_{tot} = Total pressure in the digester

C = Acid concentration

D = % total SO_2 of acid

a_1, a_2, a_3, a_4 = Regression coefficient

The S-factor is usually accurate under laboratory conditions where all the cooking factors (cooking liquor quality, wood chip quality, temperature, etc) can be controlled. In a real plant, the S-factor alone is not adequate [Haywood, 1986]. This can be confirmed because Bylund obtained a regression coefficient $R = 0.96$ using laboratory data for equation (2.19) [Bylund, 1982] and a regression coefficient $R = 0.84$ using real plant data for equation (2.20) [Haywood, 1982].

It should be noted that all those mathematical model do not have the wood chips quality as a dependent variable. It is well established that moisture, mixed wood species, contamination, chip dimension among other things have a strong impact on pulp quality [Bylund et al, 1982; Andrews et al., 1985; Parker, 1988]. If those input variables are not monitored or known, the exactitude of any mathematical model would be inaccurate.

There is no developed mathematical model that takes into account "chip size", "chips quality", and "chip moisture" and so on. As those input variables can affect the cooking process significantly, mathematical model cannot always be employed to estimate kappa numbers and "cooking time".

Artificial intelligence (AI) techniques provide methodology which can be applied to the batch pulping process to deal with such problems. The next chapter outlines the technology of AI and expert systems.

CHAPTER 3

KNOWLEDGE PROCESSING

This chapter presents some background about artificial intelligence and expert systems. It first start by giving an overview of AI techniques. The concept of "problem solving" for well-formulated and ill-structured problems is also discussed. The last part of this chapter presents a complete description of expert systems which is a branch of AI in the process industry.

3.1 OVERVIEW OF AI

The terminology "Artificial Intelligence" was first coined by John McCarthy (the creator of the computer programming language LISP) in 1956 [Johnson, 1986]. AI has been around for a long time. However, the fast growth of computer systems in recent few years has resulted in the wide applications of AI. This technique is used by many computer scientists, engineers and researchers for a wide variety of the following formal research areas;

1. Expert Systems
2. Game Playing
3. Automatic Programming
4. Neural Network
5. Speech Recognition

6. Knowledge Representation
7. Learning
8. Natural Language Processing
9. Robotics
10. Vision and Scene Understanding

The wide diversity of these various areas makes any formal definition of the term artificial intelligence difficult to justify. Perhaps the most convenient definitions of AI are along the following lines [Johnson, 1986].

The part of computer science concerned with designing intelligent computer systems, i.e., systems that exhibit characteristics that humans would consider intelligent if performed by a human;

A branch of computer science whose objective is to endow machines with reasoning and perceptual capabilities;

The area of computer science which deals with problems which are incomplete in nature, or which have indefinite solutions.

The increases in funding from both governmental and industrial sources, coupled with the steady increase in the availability of cheap computer power, have resulted in some notable achievements in specific AI areas in recent years. It is these successes that have brought some aspects of AI out of the academic research environment into the practical world of industry.

As a new advanced technology frontier, AI has been widely applied to various disciplines, including chemical industry [Stephanopoulos, 1987]. This study aims at processing non-numerical information, using heuristics and simulating human being's capability in problem-solving. In fact, a much better terminology for this field was suggested as "Complex Information Processing

System", rather than "Artificial Intelligence". Today, instead of continuing arguing which terminology is better, we concentrate on investigating the use of powerful computation technology in AI research [Rao, 1991].

The three important objectives for AI techniques are:

- i. Processing symbolic information rather than numerical data,
- ii. Using heuristics rather than algorithms, and
- iii. Simulating human behavior in problem-solving.

3.2 PROBLEM-SOLVING

Any kind of problem generally consists of 3 main components which are shown in figure 3.1. These components can be described as follows:

Initial states:

The states of a problem which consists of all the facts existing before problem-solving starts.

Goal states:

The states to be achieved after the problem is solved. In other words, it is the solution by solving the problem.

Operations:

This component is all the procedures used from the initial state to the goal state.

In term of states and operations, "problem-solving" is the process at an initial state and searching through a problem space in order to identify the sequence of operations leading to a desired goal, as shown in figure 3.1.

Generally speaking, a problem can be a *well-formulated* or *ill-structured*. Well formulated problems are those whose initial states, goal states and all legitimate operations are completely known. They can systematically generate all the problem states, and map the entire problem space. They are generally described by a mathematical equation or model. Therefore, the solution space is a subset of the problem space.

Ill-structured problems are the most common ones we are facing with. These problems have at least one of the following characteristics:

- i. Lack of one or more features of well-formulated problems,
- ii. Goals are not stated explicitly,
- iii. Problem states are not discrete,
- iv. Operations are not specified,
- v. Problem space is unbounded,
- vi. Initial states may not be fully known.

To solve ill-structured problems, different approaches than purely numerical algorithm need to be used. The following approaches are usually featured:

1. the use of experience to simplify the problem,
2. the use of heuristics for bounding search space.

The study of complex problem-solving skills becomes a matter of determining what knowledge is needed to reduce the huge problem space of an ill-structured problem to a more manageable size. In many cases, specially in the

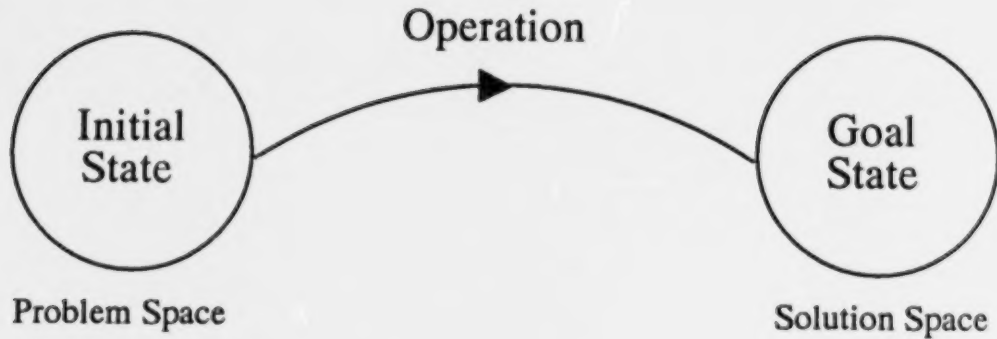


FIGURE 3.1 The three component in "Problem Solving"

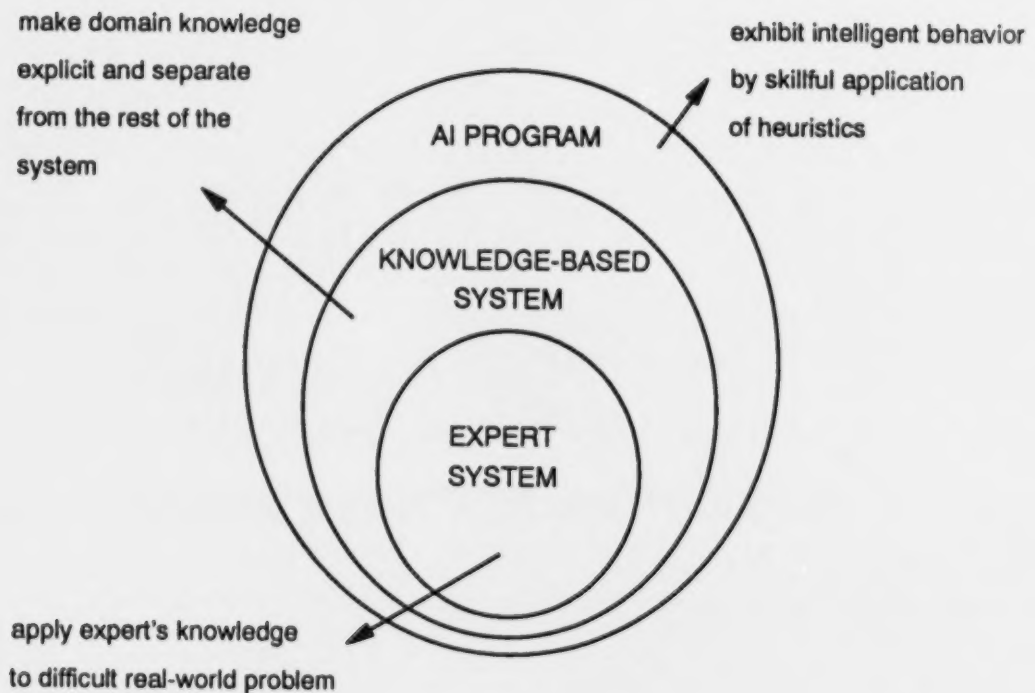


FIGURE 3.2 Relationship between "AI" and "Expert System"

manufacturing industry where most problems are not well structured, AI systems are used to reduce human problem solving by simulating human intelligent behavior.

When simulating human intelligent behavior, AI systems represent knowledge in ways more similar to human being knowledge than traditional systems. One AI technique for problem solving is *expert systems*.

3.3 EXPERT SYSTEMS

This section presents an overall description of expert systems. The development stages and the main advantages of expert systems in industry are also discussed.

3.3.1 Definition

Basically, an expert system is a computer program which represents the knowledge, judgement, and experience of an acknowledged expert in a specialized area [Egol, 1990; Finn, 1990; Azad, 1988]. Thus, the expertise of a highly skilled individual can be programmed in a computer system, and be utilized by those of less expertise [Ferrada et al, 1990; Whitehead et al., 1990; Buck, 1989]. Expert systems can be used to solved specific problems which are difficult or cannot be described by purely mathematical models or numerical algorithms.

The main difference between expert systems and conventional computing methods is that data are represented symbolically instead of numerically, and problems are solved in a symbolic reasoning fashion rather than using an algorithmic procedure [Ferrada et al, 1990; Simmonds, 1987]. As shown in figure 3.3 the architecture of an expert system consists of a "database", an

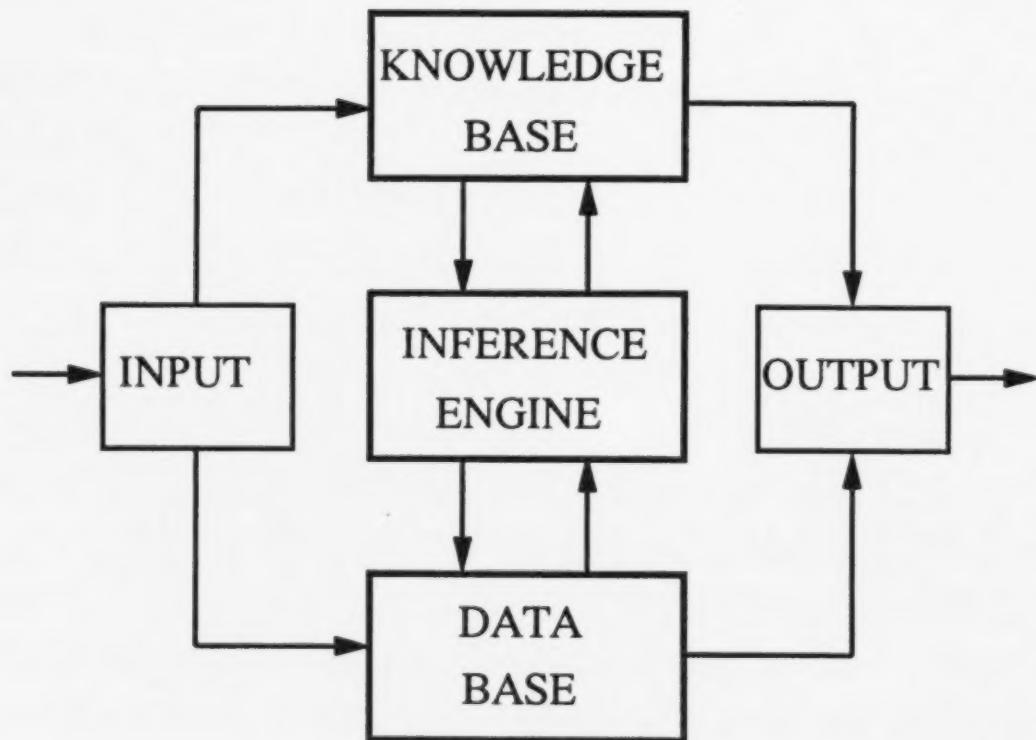


FIGURE 3.3 Architecture of "Production System"

"inference engine" and a "knowledge base". All these three components along with an input/output interface are included in a "Production System".

The database holds the knowledge that accesses to the entire system. In most cases, it contains the status of the problem to be solved. Each unit of database is an attribute-value element. Any attribute without a particular assigned value is given the default value [Corbin, 1991].

The inference engine is the reasoning mechanism whereby the rules are actuated. The application of a rule changes the system status and activates some rules while disabling others. Basically, two main reasoning mechanisms are used in inference engines. **Forward** inferencing, in which the system works forward from the facts to a solution, and **backward** inferencing, in which the system works backward from the solution to find an indication to support the solution. However, in some cases, a forward inferencing system designed by one individual could be seen as a backward inferencing system by another individual. The definition of these terms depends on how the problem is perceived. For most people, expert systems use the forward reasoning mechanism, that is, a user enters all the data about the problem into the database, and the rules are applied to reason forward from the data to find the conclusions. This type of reasoning is also called data-driven inference system.

The major difference between an inference engine and a more traditional program is its ability to manipulate abstract symbols, as opposed to numerical values. The major languages used to implement expert systems are LISP, PROLOG, and C. But almost any computer language like PASCAL and BASIC can be used.

Knowledge representation is fundamentally the heart of all AI systems, including expert systems. It defines how knowledge is captured, organized, and stored in the knowledge base of the system, accessed, and updated. The knowledge base is made up mostly of rules, which represent the domain facts and heuristics and are the rules of thumb that define the non-factual components of expertise [Gidwani, 1985]

The user interface itself is basically a typical video-workstation, with interactive, multi-window, and graphics. It tells the user where to enter the data, and how to obtain conclusions. A well designed and process adapted interface is very important [Shirley, 1987].

Knowledge Bases

The process where domain-specific knowledge is gathered from the human expert and codified into a computer system is often called knowledge engineering. It is a skilled task to extract the expert's knowledge, involving a considerable component of applied psychology.

The knowledge codified into a computer system are identified as *Public knowledge* and/or *Private knowledge*. These terms can be defined as follow:

Public knowledge:

This type of knowledge is the one obtained from public knowledge transferring techniques such as courses, literature, books, etc.

Private knowledge:

Private knowledge is the one obtained from personal experience and heuristics. Usually this type of information is the most specific and most valuable about a specific domain.

Expert systems essentially use four principle mechanisms to represent knowledge:

1st order logic

Logic expression represents factual and relational knowledge.

(i) *Propositional calculus* includes the statement that are either true or false. They are linked with AND, OR, NOT, IMPLIES, and EQUIVALENT.

(ii) *Predicate calculus* is an expression of propositional calculus. Each elementary unit is an "object". The value of fact must be either true or false. Predicate is the statement about objects.

Production rules

Production rules are used to represent relationships. Each rule has a premise (if clause) and a conclusion (then clause) [Rao, 1988]. Both the premise and conclusion can contain certainty factors. The compound statement (such as AND, OR) can be used in a rule. Rules are used to describe the logical relationships among elements of information within the domain, and used to infer values for parameters during a consultation. It specify a deduction that can be made in a particular situation. A special feature of "production rules", called *variable rule*, allows the system to substitute many different facts into the same general format, but it must have a "look-up table".

Semantic networks

Semantic network (SN) is used to describe a knowledge representation based on a network structure, which provides a means of representing facts (as nodes) and their relationships to other facts (by links) in a graphical form.

Frames

Frame is a description of an object that contains attribute for all the information associated with that object. Frame is an efficient way to represent both descriptive and procedural knowledge while increasing the modularity of the

program. It is very useful when multiple attributes are used to describe an object. Attributes may contain numerical values, default values, pointers to other frames, set of rules, and procedures to obtain values.

3.3.2 Development Procedure for Expert Systems

There are basically five stages involved in the development of expert systems. Figure 3.4 shows these stages and the interaction between each other. These stages can be described as follow:

1. Identification

In the identification stage, the problem and the goal about a specific domain are defined. This is probably the very first thing to establish in the development of any type of "problem-solving" systems. The people to participate in the development are identified in this stage. These people are the experts of the specific domain concerned, the system end users, and the system developers. A developer is usually an individual who has the knowledge in computer science and artificial intelligence.

The resources in term of developing time, computing power facilities and software tools available are also considered in this initial stages.

2. Conceptualization

The conceptualization stage consists of investigating the type of data, information, and knowledge available to the problem. The procedures involved in problem solution as well as the constraints and limitations about the problem are being evaluated in this stage.

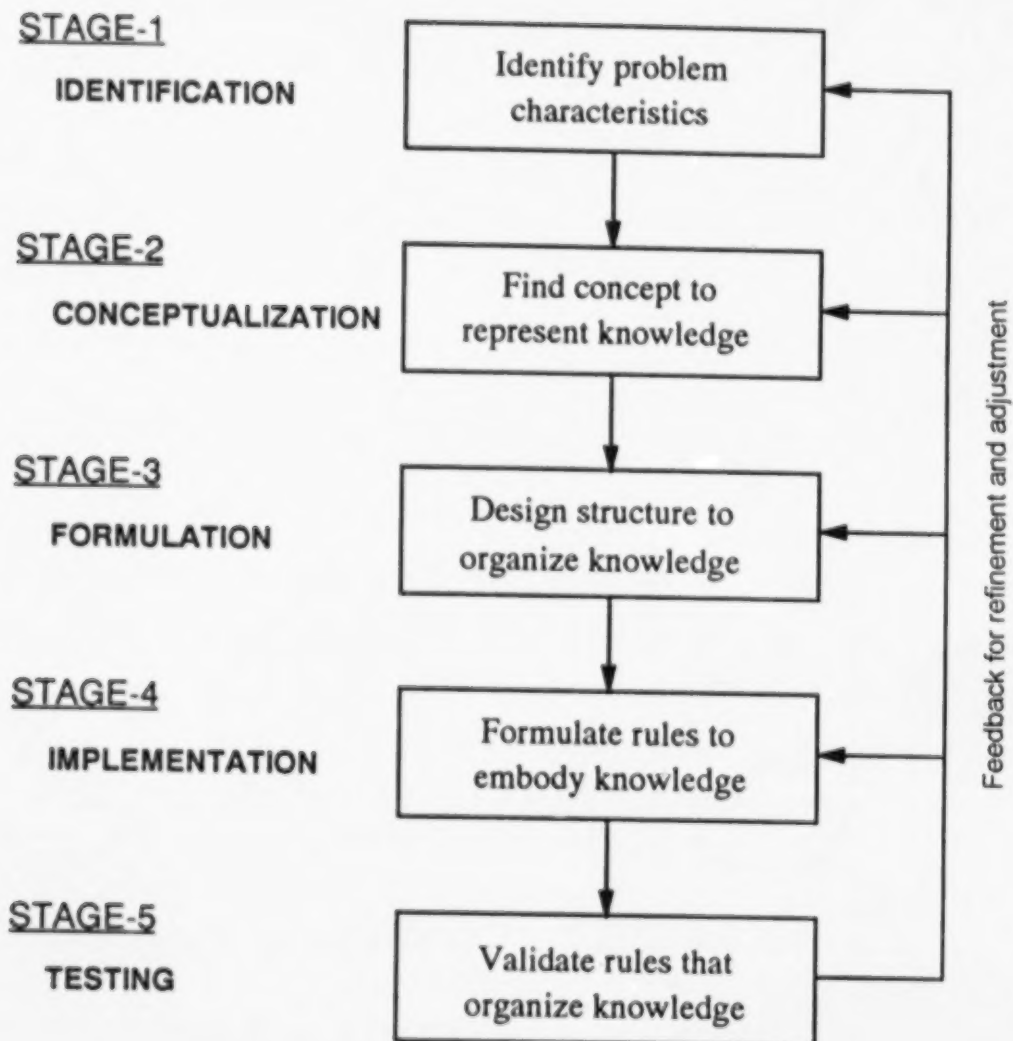


FIGURE 3.4 Standard stages for the development of "Expert Systems"

3. Formulation

The formulation stage is a process involved in mapping the key concept subproblems and information flow characteristics isolated during conceptualization stage into a more formal representation based on various tools. It consists of building knowledge representation and finding the required information to be processed by the system and the useful information to be produced by the expert system.

4. Implementation

This stage involves mapping the formulation knowledge from the previous stages into the representation framework associated with the tools chosen for this problem. The design of the user interface is also developed in this stage.

5. Testing

The testing stage evaluates the prototype system (the model system) to ensure that the system meets the requirements and the users can be well adapted through the system interface.

3.3.3 Benefits & Advantages

This sub-section outlines the main benefits and advantages from the implementation of expert systems in the manufacturing industry.

1. Levering high levels of expertise to novice levels:

Specially for training, expert system developed based on a highly skilled individual (expert) in a specific area or domain, can be accessed to other individuals of less skilled to perform the same task as the expert [Kaemmerer, 1990].

2. Improving decision-making quality and efficiency:

Expert systems can assist operators in evaluating the data to advise them how to react to changing conditions. This is very efficient when emergency actions are required[Kramer, 1987].

3. Preserving and distributing expertise:

The knowledge of a highly skilled individual can be implemented into a expert system and be preserved in the case of retirement, promotion, or resignation from the company of the individual. The information about a domain can easily be accessed to other individuals such as engineers, managers or other personnel by distributing it through an expert system.

4. Utilization of advanced computing techniques:

Artificial intelligence applications stimulate the utilization of advance computation technology.

3.3.4 AI Application in the Pulp and Paper Industry

Only a few AI applications have been implemented in the pulp and paper industry. The "SIMSMART" expert system's synthetic intelligence compiler, which creates process models from the model library, is one of them. The prototype SIMSMART was purchased by Donohue's Clermont mill. It was designed to trained operators for a newsprint paper machine [Waye, 1991].

Another prototype, called FALCON, has been tested at the University of Delaware for the chemical industry [Shirley, 1987], but no pulp and paper industrial application has yet been reported.

Some other papers about AI technology have been published in pulp and paper journals [Lukas at al., 1989; Waye, 1987], but based on our literature search, very few publications discussed the actual implementation of AI or expert systems in the pulp and paper industry.

CHAPTER 4

PROBLEM DEFINITION

The industrial pulping process investigated in this project is one of Fraser's sulphite mills. Fraser Inc., which is located in Edmundston, New-Brunswick, operates eight batch digesters in its sulphite mill. The size of these digesters ranges from 223 m³ to 234 m³. The entire mill produces approximately 500 tonnes of unbleached chemical pulp per day. Digester #4 is the unit used in our case study.

This pulping process is fairly complicated. Its operation and related problems are described in this chapter, starting with a description of all the phases of the process operation, followed by a clarification of the problems related to the process operation/monitoring.

4.1 PROCESS OPERATION

This section deals with the several phases of operation of the cooking cycle as practiced at the Edmundston sulphite mill. There are nine major operations in the cooking cycle, starting with the introduction of the wood chips into the digester all the way to the dumping stage when the cooking vessel is emptied [Smith, 1983].

A plot of the temperatures and pressure versus time for a cooking cycle is shown in Figure 4.1. This figure shows two temperature curves (one for the region above strainer plates, and one for the region below strainer plates), and one pressure curve. As all these three curves are plotted on a real-time mechanical recorder strip chart, a physical lag between the pens is required to prevent inter-pen obstruction. This lag represents a 10 minutes interval between each plotting pen.

Figure 4.1 also reflects all the required operation for a complete batch cooking cycle. These required operation procedures are described in the following lines.

Operation 1: Chip Filling

Wood chips from a woodroom and/or from a chip pile are transported to the chip bins located above the digesters' house. When a digester is ready for a new cycle, its top cover is removed. A chute is moved into place and ready to pass the wood chips into the digester. It is important that the dump valve located at the bottom of the digester is completely closed prior to the loading operation. During "chip filling", steam of 345 kPa is injected tangentially near the top of the digester thus creating rotating motion of the chips to produce a denser packing per batch. In this manner, the amount of chips in the digester is increased, thus increasing the yield. The average duration for the "steam packing" process is approximately 25 minutes.

Operation 2: Capping

Once the digester is filled with wood chips, the chute is removed. A gasket is placed upon the digester's opening, the top cover is replaced, and secured

with many bolts. Under normal conditions, the moisture content of the wood chips loaded into the digester is approximately 51 %.

Operation 3: Acid Filling

After the digester has been completely sealed, the vessel is ready for the "acid filling" phase of operation. The operator presses the button on the temperature programmer to start the cooking process. Upon pressing the "start button", the digester's vent is automatically opened, the acid feed valve is also opened, and the cooking acid starts entering the digester. The cooking acid is transferred from the accumulator at a rate of approximately 1250 m³ per hr. As this operation occurs, a cooking liquor sample is taken by the operator, and its quality (total %SO₂, free %SO₂, combined %SO₂, and pH) is analyzed. Only one digester can be filled at a time. During "acid filling", the acid enters the strainer plates, thus flushes these plates. After a pre-set number of cubic meters of acid has entered the vessel, the circulation pump starts automatically.

Operation 4: Indirect Steaming

Each digester is equipped with its steam fed heat exchanger in which the circulating cooking liquor is heated. A mechanical temperature programmer is used to increase and control the temperature of the digester's content, by manipulating the steam going to the liquor heater. The programmer consists of a drum equipped with a temperature-time curve. During the cooking cycle, the drum rotates, and a temperature set-point needle follows the cooking curve.

Operation 5: Pressure Impregnation

The "pressure impregnation" operation consists of forcing the cooking acid to penetrate the wood chips. During the "rising time" (while increasing the temperature) of the digester, an impregnation pump and other equipments are

used to achieve the desired cycling pressure as shown in figure 4.1. The objective of this operation is to widen the membrane pores in the wood chips, and thus facilitate the penetration of the cooking acid into the chips. The impregnation operation is pre-programmed.

Operation 6: Side Relief

The purpose of the operation of "side relief" is to make the "cooking acid strength (total SO₂) / air dry lbs of wood" ratio equal to 18% by removing a calculated amount of cooking liquor from the digester. This operation also allows the formation of a vapor space inside the vessel to safely maintain a high pressure which is required for delignification. "Side relief" is programmed to occur when the digester reaches 110°C. The "side relief" liquor returns to the accumulator by the circulating pump.

Operation 7: Drain Down

After the digester has been operated for a certain period of time, and the operator has estimated that the desired pulp quality has been reached, cooking liquor is drained from the digester. The cooking liquor removed is called the "drain down". The purpose of this operation is not only to reduce the total pressure in the vessel, but also to make room for the quenching liquor, which is introduced in the next phase of operation. A well judged "drain down" operation is the most important part because it results in good quality pulp.

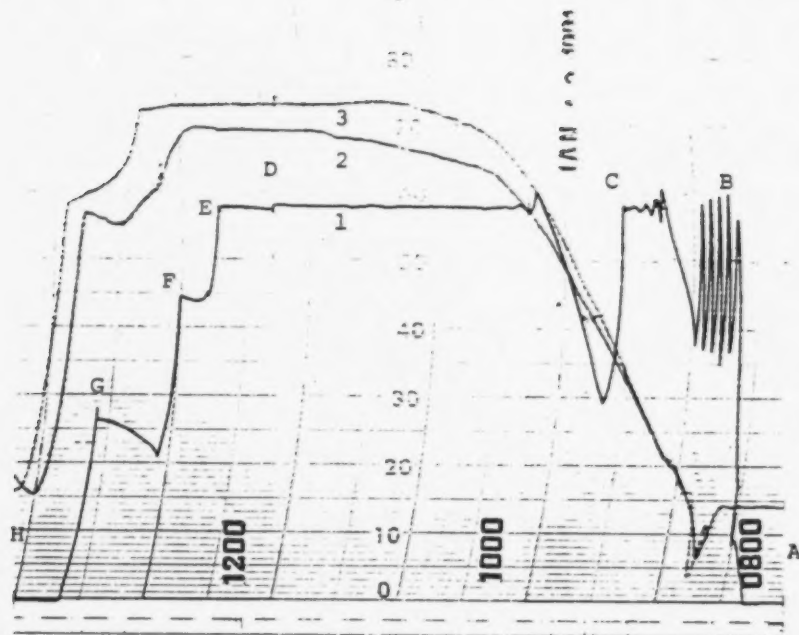
Operation 8: Quenching

During the quenching phase, cold "weak liquor" is introduced into the digester to further reduce pressure and temperature in the cooking vessel. This weak liquor has practically no chemical strength; and comes from the washing and screening process operation of the plant. For a short period of time, it is

introduced into the cooking vessel through two different locations. First it is added at the top and the bottom of the digester through the circulation line, then it is only introduced at the top of the digester. The main objective of quenching the digester content is to stop the cooking process to prevent dissolving of the cellulose portion of the wood chips, thus preventing loss of yield and also possible loss in pulp brightness and pulp strength.

Operation 9: Flushing & Dumping

After the digester has been quenched, the "dump valve" is opened and the vessel is emptied. The pulp passes through several processes for further refinement including "rock trap", "defiberizer", "washing & screening", "bleachery", "paper machines" [see Figure 2.1]. If the digester is not emptied immediately after the quenching phase, pulp brightness, pulp yield, and pulp quality would suffer. Such a delay would be due to a shut down somewhere down stream of the processing line. During the "dumping" operation, the consistency of the pulp is estimated at 4% fibers, and the stock temperature is under 95 °C.



CURVES

- 1- TOTAL PRESSURE
- 2- TEMPERATURE ABOVE STRAINER (Bulk Temperature)
- 3- TEMPERATURE BELOW STRAINER

STEPS

- A- Digester is full of wood chips and liquor is being pumped
- B- Impregnation cycles
- C- Side relief
- D- Cooking process at operating temperature and pressure
- E- Drain down - (target kappa is estimated to be achieved)
- F- Quenching (addition of cool weak liquor)
- G- Blowing/dumping
- H- End of the whole cycle

FIGURE 4.1 "Temperature & Pressure" curves of a cooking cycle

4.2 PROCESS ANALYSIS

This section presents an analysis of the main problems related to the performance of the process operation.

4.2.1 Objectives

Digesters

The digesters cook ("digest") wood chips to a specified degree of delignification (*Kappa Number*). That is, it dissolves and removes the lignin content from wood chips to a pre-specified extent so that only a permissible amount of lignin remains in the pulp. To achieve this objective, a certain chip quality, cooking acid quality, and operating conditions (temperature, pressure, and cooking time) are required. Whenever one of these variables is out of its required range, the operator usually changes other variables for compensation to achieve operation objective.

Operators

The chemical process operators monitor and control a process to produce products with the desired quality. Their objective is to ensure that the process meets its goal, using their knowledge and skills as well as experience which they have acquired during their training and confirmed through years of operation.

In this study, the operators' objective is to control the digester to produce the best pulp quality based on given raw materials, by choosing the appropriate operating conditions (cooking time). The operator must decide when to terminate the process by pressing the "drain down" button, so that good pulp quality can be obtained in the digester at the end of the process.

4.2.2 Problems

Unfortunately, the kappa number cannot be measured on-line, it can only be determined after the process has been completed through a laboratory test. The lignin content in the wood (pulp) cannot be measured during the batch cooking cycle. As the process operates at very high temperature and pressure, it is not practical to get a pulp sample from the reactor vessel. As described in chapter 2, the kappa number is determined by a laboratory test involving titration performed on the pulp sample. At Fraser, this test is done locally in the sulphite mill laboratory.

The pulp sample is taken from the dump line during the "dumping" operation of the digester after the cooking cycle is completed. The kappa number test is then performed on the sample and the results are entered on the digester log sheet. These values are used as a guide by the operators at the bleachery plant for the bleaching process. The "target" kappa number at the Fraser's sulphite mill is 28. A value between 26 and 30 is considered satisfactory. In the case that the laboratory test shows a kappa number higher than 30, indicating a high lignin content still remaining in the pulp, the bleachery operator needs to increase the rate of chlorine/chlorine dioxide for pulp bleaching.

A low kappa number of the collected sample (i.e. below 26) indicates loss of pulp strength, pulp brightness, and pulp yield for that particular "batch cook". Occasionally, the quality is so bad that the pulp cannot be used for paper making. It can only be used at the "box board" mill, for the production of card board, thus resulting in the production of a low grade paper product from pulp made by a high grade pulping process

At Fraser's sulphite mill, under normal conditions, a cooking time of 3:45-3:55 hour is required for the batch process to produce a pulp of satisfactory quality (i.e. a kappa number between 26 and 30). Another difficulty is that the raw material qualities ("cooking acid" and "wood chips") that are charged into the digester are not always as anticipated. The cooking acid is produced in an upstream "Recovery" process, and its quality varies significantly, affecting the cooking time (i.e. reaction time). The chips quality, which sometimes differs in term of moisture content, species, and age; produces the similar effect on "cooking time". For example, a certain quality of raw material will require a cooking time of 3:35 hour for the digester to achieve the goal its objective, whereas under different circumstance, the batch cycle would required 4 hour of cooking time to obtain the satisfactory pulp quality.

It is the digesters' operator who must decide when to terminate the cycle during the cooking process. The only assistance that is currently being used by the operators to recognize the right moment to initiate "drain down" of a digester, is by testing the total % SO_2 of the cooking liquor. During a cooking cycle, the operator tests the total % SO_2 of the acid, and a value between 0.8 to 0.9% is taken as an indication that a satisfactory kappa number will be reached, and the "drain down" button should be pressed.

Figure 4.2 is a bar graph showing the frequency of the "batch cook" kappa numbers for the month of July, 1990 during my first field interview at the mill.

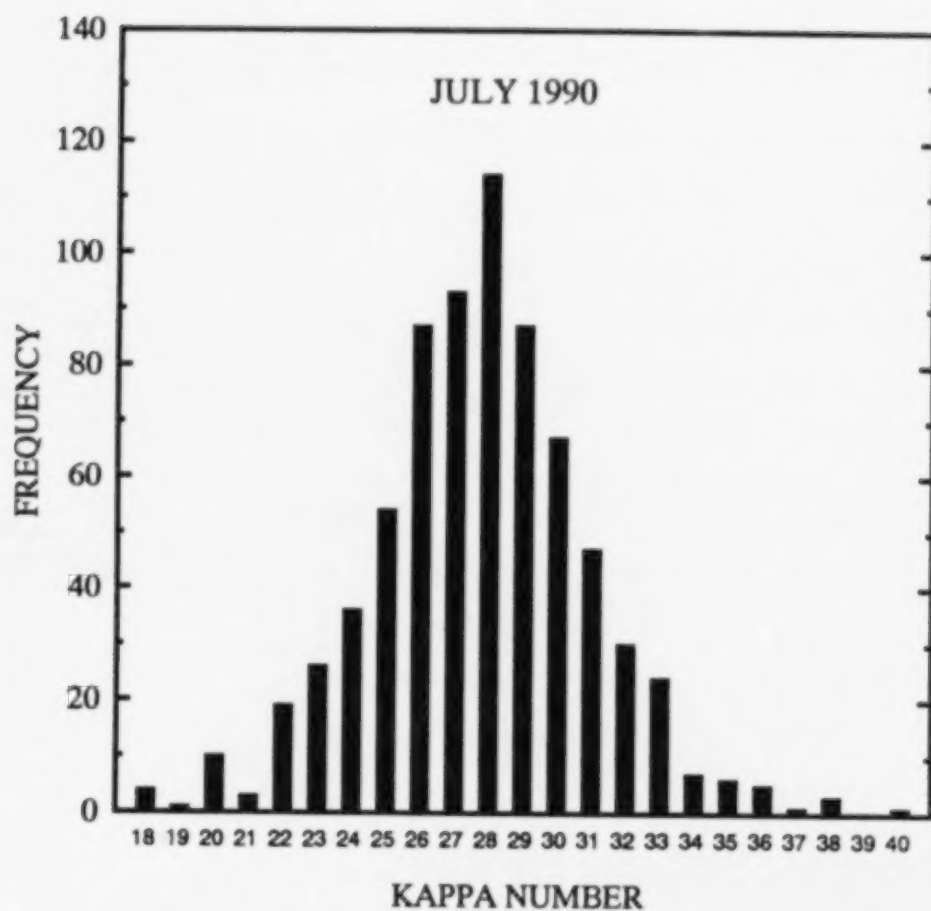


FIGURE 4.2 Frequency of Kappa Number

The figure shows that out of 725 batches produced during that month, only 114 (15.7%) had reached the target kappa number (i.e. 28). Other batch operation (61.8%) were within the desired satisfactory limits (i.e. between 26 and 30). One hundred and fifty three (21.1%) were overcooked and one hundred and twenty four (17.1%) were under cooked. These figures were obtained from the daily log sheets at the sulphite mill and are taken as representing the annual average.

Table 4.1 shows that 277 cooks (38.2%) produced during July 1990 were considered of poor quality, indicating that the current method for the pulp quality prediction is not totally satisfactory.

Table 4.1: Performance of the cooking process operation

TOTAL BATCH COOKS	725	KAPPA NUMBER	100%
Batches of Best Quality	114	28	15.7%
Batches of Satisfactory Quality	334	26-27 29-30	46.1%
Batches of Poor Quality	under 124	>30	21.1%
	over 153	<26	17.1%

4.3 PROBLEM SOLVING

As the problems have been explained in the previous section, a strategy to solve these problems is presented in this section.

4.3.1 Strategy

Many factors can be the cause why 38.2% (on an average) of the kappa numbers do not fall within satisfactory limits. One of the main reason can be

attributed to a wrong interpretation of the process information/data by a human operator. One human operator is responsible for all eight digesters during a 12 hours shift. Even though some experienced operators have many years of operation experience and know the process in detail, they are still inefficient to quickly recall the effect of data for a digester and other data for another digester. In some cases, they are getting confused during the operation.

Again, the main criteria to produce a desired pulp quality is to choose the right "cooking time" and maintain the right operating condition. Cooking time is a function of the raw materials quality (wood chips & cooking liquor) and the operating conditions (temperature and pressure).

The strategy is to develop an intelligent advisory system that suggests the proper "cooking times" based on the available information concerning the process, and monitoring the cooking process to estimate kappa numbers. By investigating the effect of all the available process quality inputs on the cooking time (private knowledge), a computer program can be developed using artificial intelligence techniques to quickly processes the knowledge and provides decision support. The private knowledge is obtained from the experts (operators), and can be coded into the system. A mathematical cooking model developed based on the "S-factor" (public knowledge) can also be implemented in such a computer system, and can be used for supervisory control.

4.3.2 Implementation Procedure

In the implementation of expert system for the operation and monitoring of industrial processes, three practical phases are suggested:

Phase 1: Off-line system

The objective at this phase is to organize the system and codify the knowledge, and to evaluate the expert system in an off-line environment, such as design and training as well as off-line supervisor. A system at this phase is basically a simulation where the entry are performed by the user/operator over a keyboard. Such system is excellent for training because its a highly interactive system. Most academic expert systems are mainly developed at this phase. Its main function is to provide decision support to human operator based on real-time information.

Phase 2: On-line supervisory system

The objective of this phase is to evaluate the interface between both human operator and hardware instrumentation. At this phase, the expert system is physically connected to the actual process via instrumentations, and the process information are fully or partially read by the expert system.

Phase 3: On-line & closed loop system

A close loop expert system directly reads the inputs from the process and sends its outputs (action) back to the process. However, keeping the process operator outside the loop is almost inescapable. In most cases, the objective of these system is to reduce the operators' tasks and improve operation safety and efficiency.

The expert system developed for this project was implemented at phase 1 and phase 2. In this case, the system was not physically connected to the process.

Figure 4.3 is a diagram of the possible channels of information transfer. The upward-pointing arrows indicate the use of sensors and the downward-pointing arrows the use of actuators.

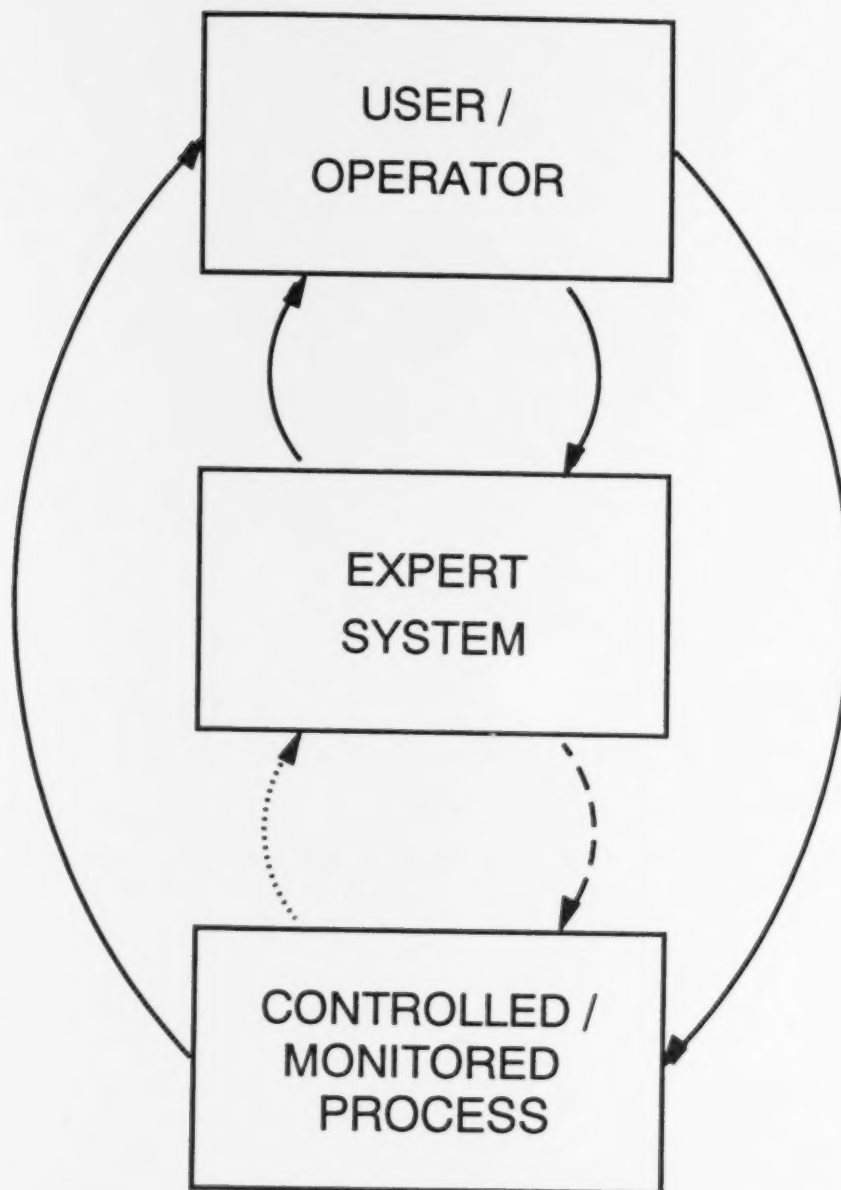


FIGURE 3.5 Information Flow of an Overall System

CHAPTER 5

KNOWLEDGE ACQUISITION AND REPRESENTATION

In chapter 5, the knowledge acquired from the field experts for the construction of the "Intelligent Operation Support System" (IOSS) is presented. An outline of this chapter is as follows. The chapter begins with a section describing the objectives of each component. In section 2, the developer's background related to the pulp and paper industry, and about the interviews with the process operators (experts) are presented. The last two sections of this chapter present the "qualitative reasoning" and the "quantitative computation" which have been acquired and implemented into IOSS.

5.1 OBJECTIVES

It is important for the readers to understand the objectives and the functions of each component involved in the development of IOSS. Figure 5.1 exhibits the flow of information and knowledge between each component. IOSS has been constructed based on that structure.

To briefly explain figure 5.1, the operators learn the process operations (i.e. private knowledge) through experience. The developer gains the private

knowledge through series of interviews with the operators (i.e. experts), then builds IOSS.

As mentioned in chapter 4, the digester's objective is to produce the chemical pulp of a specific quality. The operators are responsible for monitoring the digester to accomplish operation objectives.

Our first task is to learn the process operation. It is important for the developer to understand in detail the digester's operation and the related problems before starting the construction of IOSS. The main objective for the developer is to acquire the valuable knowledge that has been used in the digesters' operation to produce a satisfactory product. Then, the acquired knowledge is implemented into IOSS so that the knowledge can be easily transferred to the users.

The objectives of the intelligent operation support system (IOSS) is to give assistance to operators for the production of good quality pulp. The more accurate and the better organization of the knowledge are, the more efficient of the IOSS is.

5.2 KNOWLEDGE ACQUISITION

As this project required the specific knowledge and information about the batch digester operation, two field trips to Fraser and several field visits to local pulping industry plants were conducted during my graduate studies at the University of Alberta. The duration for each of these visits at Fraser was approximately 6 weeks beginning in July 1990. During this period, the problems concerning the operation of the digesters were defined. All available process quality variables were investigated and studied. Based on the first visit at the

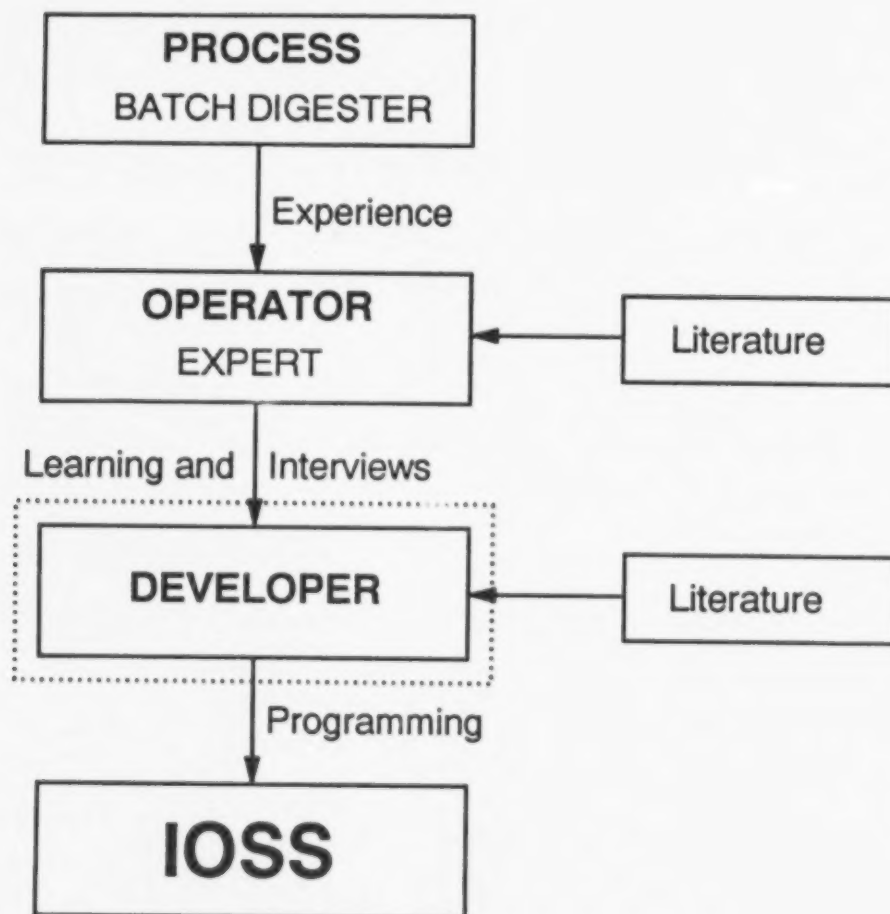


FIGURE 5.1 Flow of knowledge for the development of IOSS

plant, a knowledge representation model was built. This model was then improved and approved by the plant operation consulting experts during my second visit at the mill between December 1990 and January 1991.

Most of the knowledge acquired during these two 6 week terms are classified as "private knowledge". This knowledge, which is mainly acquired from the operators (experts), is the most important and valuable information. Private knowledge is very specific about the process behavior and cannot be obtained from the literature. A total of four operators and one superintendent were interviewed for private knowledge acquisition.

During each interview, it was important to tell the experts about this project. Some operators were concerned about the implementation of such computer system in the plant for the reason of the job securities. Technically, IOSS is a computer program that solves problems which only experts can do. It is important to tell operators that humans are much better than computers at reasoning level [Surko, 1989], so that the operators do not need to have the feeling that they are selling out their knowledge and experiences to the management and helping to automate them out of job. Finally, operators understood that the objective of the IOSS is to help them in "decision-making" for the production of good quality pulp, and should not worry about losing their jobs.

As the author already knew all the experts interviewed due to past summer employments with the company (e.i. Fraser Inc.), no major communications or cooperation difficulties appeared.

5.3 QUALITATIVE KNOWLEDGE REPRESENTATION

This section presents the "private knowledge" acquired from the operators and implemented into the IOSS.

Once the process operation and related problems were studied, all the available quality inputs and outputs were investigated. These investigated process quality variables can be classified as follows:

Process Quality Inputs

WOOD CHIPS

Chips Quality
Chips Size
Chips Load

COOKING ACID

Total % SO₂
Free % SO₂
Combined %SO₂
pH

OPERATING CONDITIONS

Temperature
Pressure
Cooking Time

Process Quality Outputs

PULP QUALITY

Kappa Number
Pulp Brightness

Table 5.1 shows the desired conditions for all process variables. If one process quality input is out of its desired conditions, the desired quality pulp would not be achieved. The impacts of the deviation of each process input on the pulping process behavior are described in the remaining of this section. Figures 5.2 through 5.9 are also included at the end of this section to demonstrate the qualitative knowledge representation of all investigated quality inputs for the IOSS.

Chip Load

Generally, a batch digester is filled with wood chips up to the top at a specific level. Weight cells installed at the bottom of the digester are used to measure the load of chips. The weight cells are calibrated to provide values between 0% to 100%. A value between 35%-40% gauge indicates an optimum chip load. A value higher or lower than the desired condition indicates that the moisture content is higher or lower, respectively. If a low moisture content in the wood chips is fed into the vessel, less cooking time would be required for the batch cycle because it is easier for the acid to penetrate the chips and to chemically attack the lignin. Also under low chip moisture conditions, less steam is required to heat up the system, and a higher brightness of the final pulp is usually resulted.

In the event of a high moisture content, more cooking time would be required. First of all, the acid concentration would be reduced. Secondly it would take more time for the acid to penetrate the chips. A higher steam rate is also required under high moisture chips conditions.

Chip Quality

This input identifies the wood chips as old chips or the type of wood blend (hardwood or softwood). Old chips are generally easy to digest but produce dark pulp (low pulp brightness). When the chips loaded into the digester are considered "old", less cooking time is required as well as a lower steam demand.

TABLE 5.1: Desired conditions for pulping quality variables

Quality Inputs	Desired Conditions
LOAD of wood chips	35 % - 40 % gauge
SIZE of wood chips	1"-3" x 1"-3" x 1/4"-1/2"
QUALITY of wood chips	Good quality (less than 6 weeks old) 90% softwood - 10% hardwood
TOTAL % SO ₂ of initial cooking acid	5.45% - 6.15%
COMBINED % SO ₂ of initial cooking acid	2.65% - 3.00%
FREE % SO ₂ of initial cooking acid	2.80% - 3.15%
pH of initial cooking acid	3.4 - 3.6
Operating TEMPERATURE	162 - 165 °C
Operating PRESSURE	675 - 680 - 685 kPa
COOKING TIME of batch cycle	3:45 - 3:55 hour
Quality Outputs	Desired Conditions
Kappa Number	26 - ~~28~~ - 30
Pulp Brightness	62 - 68

At Fraser, the blend of wood chips under desired condition is approximately 10% hardwood and 90% softwood. If more than 10% hardwood is used in the digester, the cooking time and the pulp strength will be affected. A high hardwood content (i.e. >10%) in the batch cooking results in a shorter cooking time. A higher pulp brightness and a lower pulp strength is also obtained as the final results. Because hardwood is made of short fibers, if the chip blend contains more than 10% hardwood, problems would eventually occur at the washing and screening system.

If the wood chips fed into the digester are both "old chips" and "more than 10% hardwood" at the same time, a much less cooking time would be required. The pulp brightness is almost unpredictable under this situation, and the strength of the pulp would be lower. On the other hand, less steam would be required to heat-up the system.

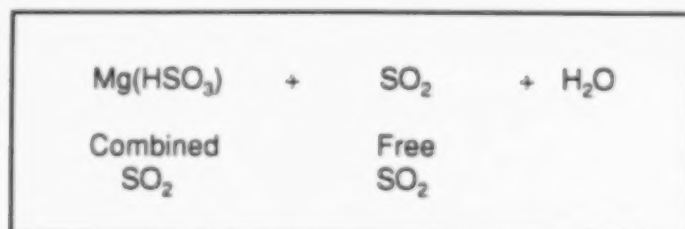
Chip Size

The standard size of a single wood chip is approximately 1"-3" x 1"-3" x 1/4" - 1/2" in dimension. If the average dimension of the chip is much different than the standard size, a different cooking time will be required for the batch cycle to achieved the right pulp quality.

In the case where the size of chips loaded into the digester are larger than the standard one, which usually occurs when the chip screening system is malfunctioning, more cooking time would be required because it take more time for the acid to penetrate the whole chips for delignification. If the chips are too small (i.e. under standard size), less cooking time would be required and a lower pulp brightness would be obtained in the final pulp.

Total SO₂

The cooking acid that is fed into the digester consists of the following major components:



The total SO_2 consists of both the combined SO_2 and the free SO_2 together. The concentration of these components are measured in percentage sulfur dioxide. The higher the total SO_2 percentage of the cooking acid is, the less the cooking time is required. The final pulp brightness can also be lower depending on the free SO_2 and the combined SO_2 ratio. If the total SO_2 percentage is low, more cooking time will be required to obtain a desired pulp quality.

Free SO₂ and combined SO₂

The normal condition for the free SO_2 and combined SO_2 of the cooking acid is when the difference between the free SO_2 percentage and the combined SO_2 percentage is 0.15% to 0.20%. In the case when the free SO_2 minus the combined SO_2 is less than the desired condition, which indicates a low acid acidity (high pH), more cooking time would be required. If the free SO_2 minus the combined SO_2 is higher than the desired conditions, which causes a high acid acidity (low pH), less cooking time would be required.

pH

For the pH of the cooking acid, a low value indicates a higher liquor acidity resulting in less cooking time requirement. Under high cooking acid acidity, the final pulp brightness produced is usually lower. If the pH of acid is high, indicating a lower cooking acid acidity, more cooking time would be required, and the pulp brightness would not be affected.

Operating Pressure

The operating pressure of the cooking cycle is the total pressure inside the digester during the digesting process at operating temperature. As the vessel is completely sealed, the pressure is raised by raising the temperature. The operating pressure is set-up at 680 kPa and maintained by means of a pressure relief valve. In some situations, when the pressure relief valve does not work properly or when the pressure set-point is incorrectly adjusted, a different operating pressure is obtained. If the operating pressure is lower than the desired condition, more cooking time is required. In the situation when a higher operating pressure is used, less cooking time would be required and a lower pulp brightness would be obtained in the final pulp product.

Operating Temperature

The effect of the operating temperature behaves in the same way as the operating pressure. A low operating temperature results in more cooking time requirements, and a high operating temperature results in less cooking time requirements. A high operating temperature also produces the low pulp brightness.

The reason why the operating temperature is outside its desired conditions is mainly due to the steam quality and the malfunctioning of the steam control valve.

Table 5.2 shows the magnitude of the required cooking time for different process quality inputs conditions. All information was acquired from the operators. Because the normal cooking time is within a 10 minute range (i.e. between 3:45 hours to 3:55 hours), a minimum of 10 minutes ranges for the cooking time is used in order to classify the different inputs conditions. When more than one quality inputs are outside their desired condition, these ranges can be compiled together to derive a coupled cooking time (i.e. a near optimum cooking time range) using human expertise.

Figures 5.2 through 5.9 present the qualitative knowledge organization for each of these qualitative variables, respectively.

TABLE 5.2: Description of each situation

Individual Inputs Quality	Description of Each Situation (RULE)	Average Cooking Time (based on individual inputs)
Wood Chips Load	22 % gauge to 29 % gauge 30 % gauge to 34 % gauge 35 % gauge to 40 % gauge 41 % gauge to 46 % gauge 47 % gauge to 55 % gauge	3:20 hr to 3:30 hr 3:30 hr to 3:40 hr 3:45 hr to 3:55 hr 4:00 hr to 4:10 hr 4:10 hr to 4:20 hr
Wood Chips Quality	Good quality Old chips More than 10% hardwood "Old chips" & "more than 10% hardwood"	3:45 hr to 3:55 hr 3:25 hr to 3:35 hr 3:25 hr to 3:35 hr 3:15 hr to 3:35 hr
Wood Chips Size	Standard Size Over Standard Size Under Standard Size	3:45 hr to 3:55 hr 4:05 hr to 4:15 hr 3:25 hr to 3:35 hr
Total % SO ₂ of Initial Cooking Liquor	4.00 % to 4.80 % 4.81 % to 5.44 % 5.45 % to 6.15 % 6.16 % to 6.50 % 6.51 % to 7.40 %	4:05 hr to 4:15 hr 3:55 hr to 4:05 hr 3:45 hr to 3:55 hr 3:35 hr to 3:45 hr 3:25 hr to 3:35 hr
pH of Initial Cooking Liquor	2.5 to 2.9 3.0 to 3.3 3.4 to 3.6 3.7 to 4.0 4.1 to 4.6	3:25 hr to 3:35 hr 3:35 hr to 3:45 hr 3:45 hr to 3:55 hr 3:55 hr to 4:05 hr 4:05 hr to 4:15 hr
Operating Temperature	154 °C to 157 °C 158 °C to 161 °C 162 °C to 165 °C 166 °C to 169 °C 170 °C to 175 °C	4:10 hr to 4:20 hr 4:00 hr to 4:10 hr 3:45 hr to 3:55 hr 3:30 hr to 3:40 hr 3:20 hr to 3:30 hr
Operating Pressure	575 kPa to 624 kPa 625 kPa to 674 kPa 675 kPa to 685 kPa 686 kPa to 735 kPa 736 kPa to 785 kPa	4:05 hr to 4:15 hr 3:55 hr to 4:05 hr 3:45 hr to 3:55 hr 3:35 hr to 3:45 hr 3:25 hr to 3:35 hr

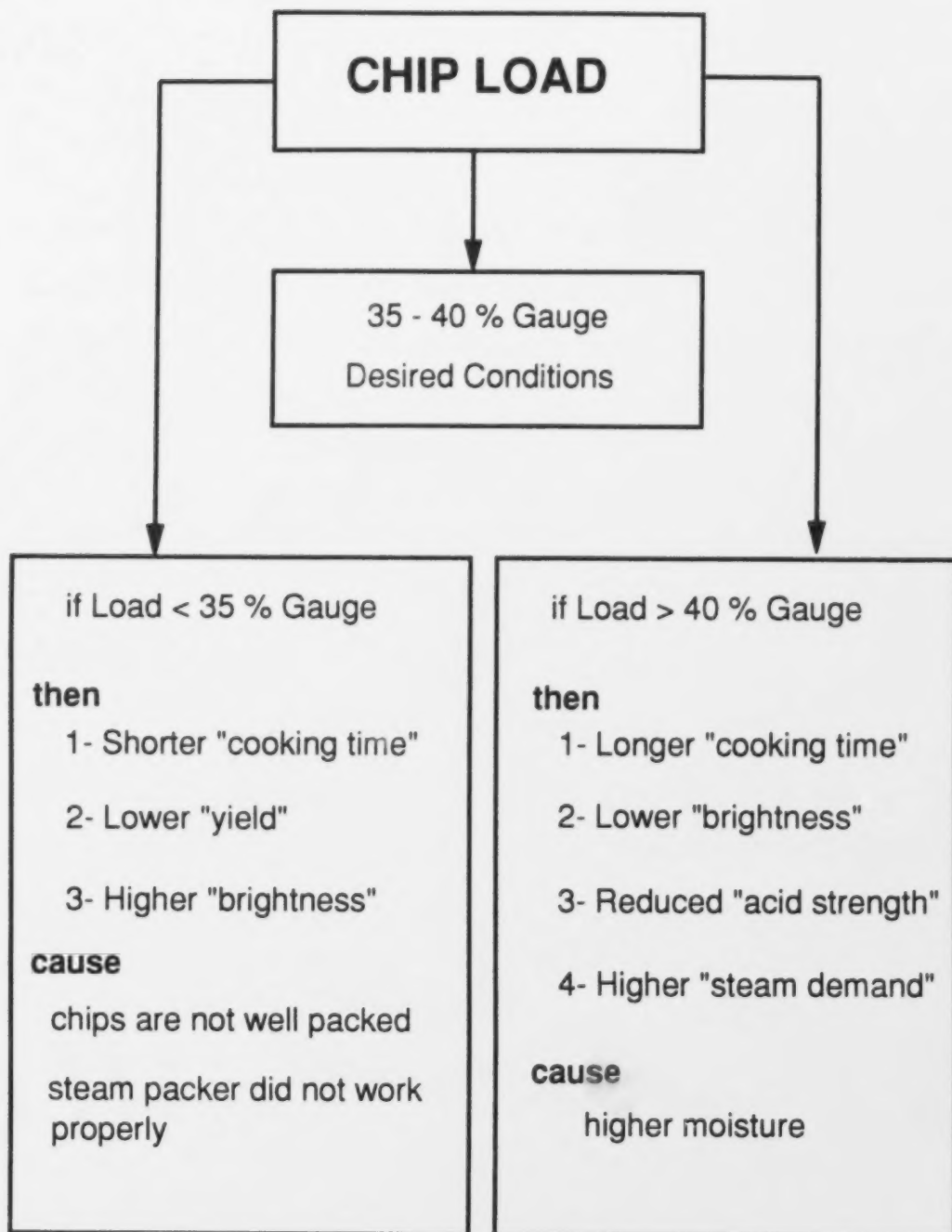


FIGURE 5.2 Qualitative knowledge for "Chip Load"

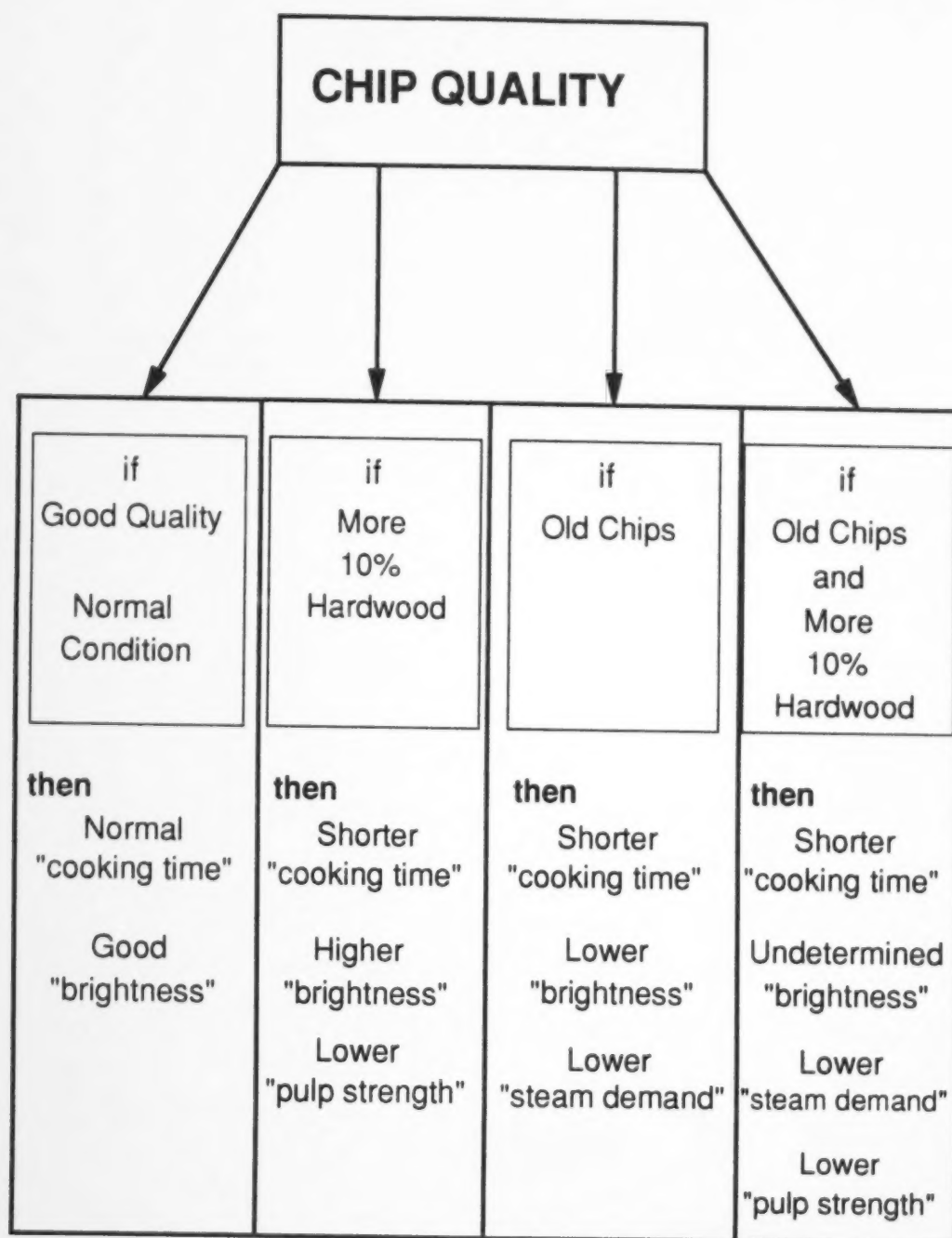


FIGURE 5.3 Qualitative knowledge for "Chip Quality"

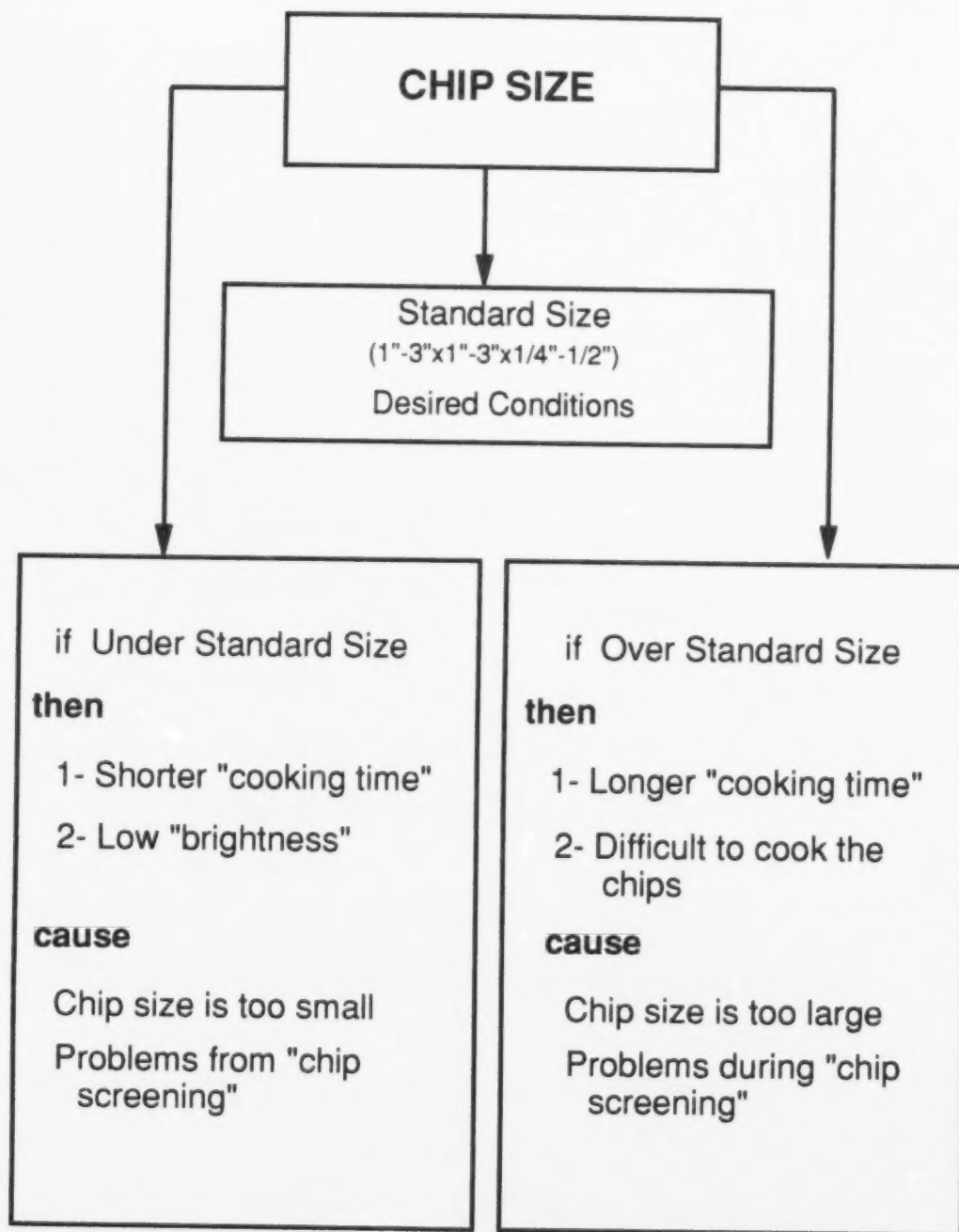


FIGURE 5.4 Qualitative knowledge for "Chip Size"

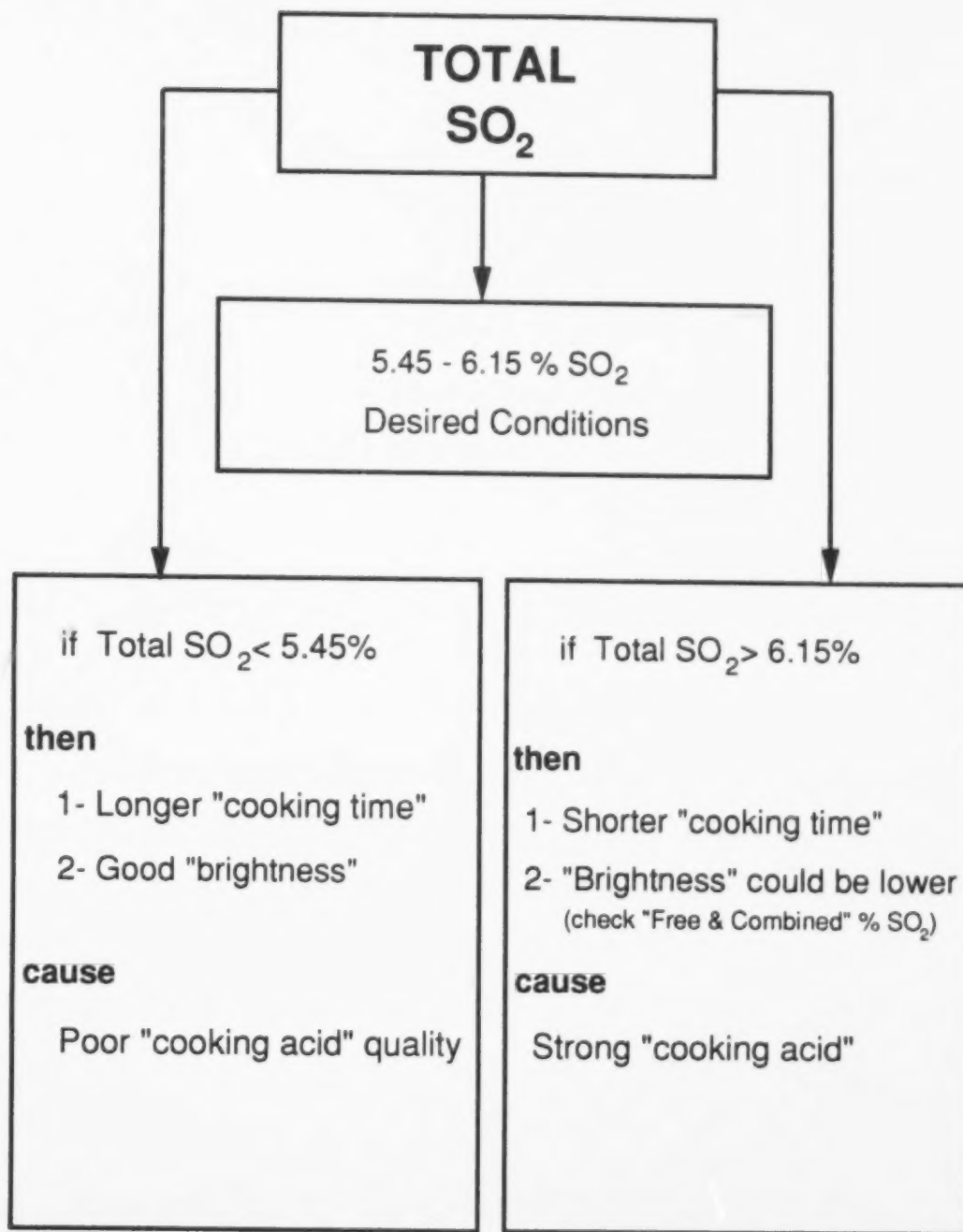


FIGURE 5.5 Qualitative knowledge for "Total %SO₂"

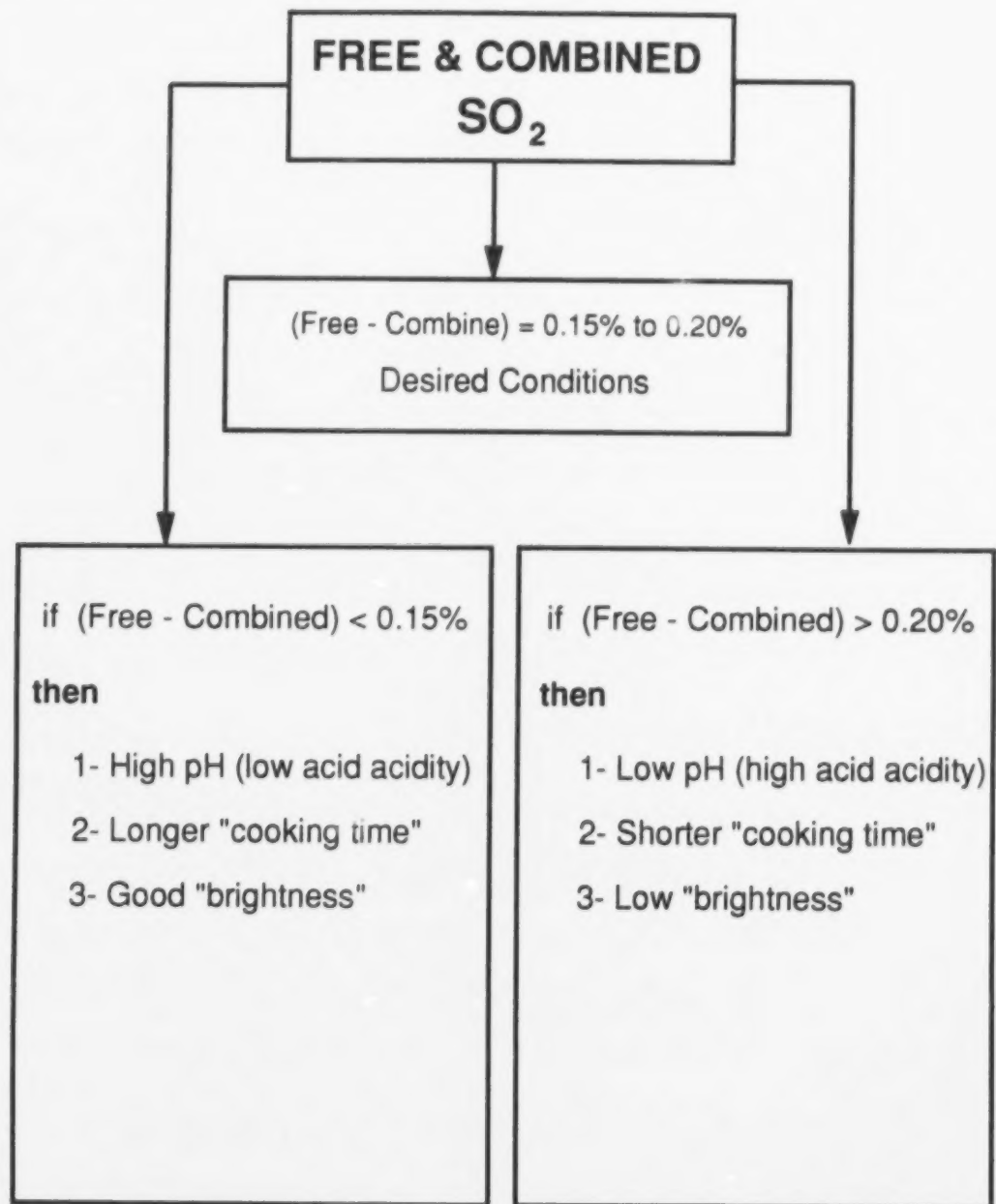


FIGURE 5.6 Qualitative knowledge for "Free & Combined %SO₂"

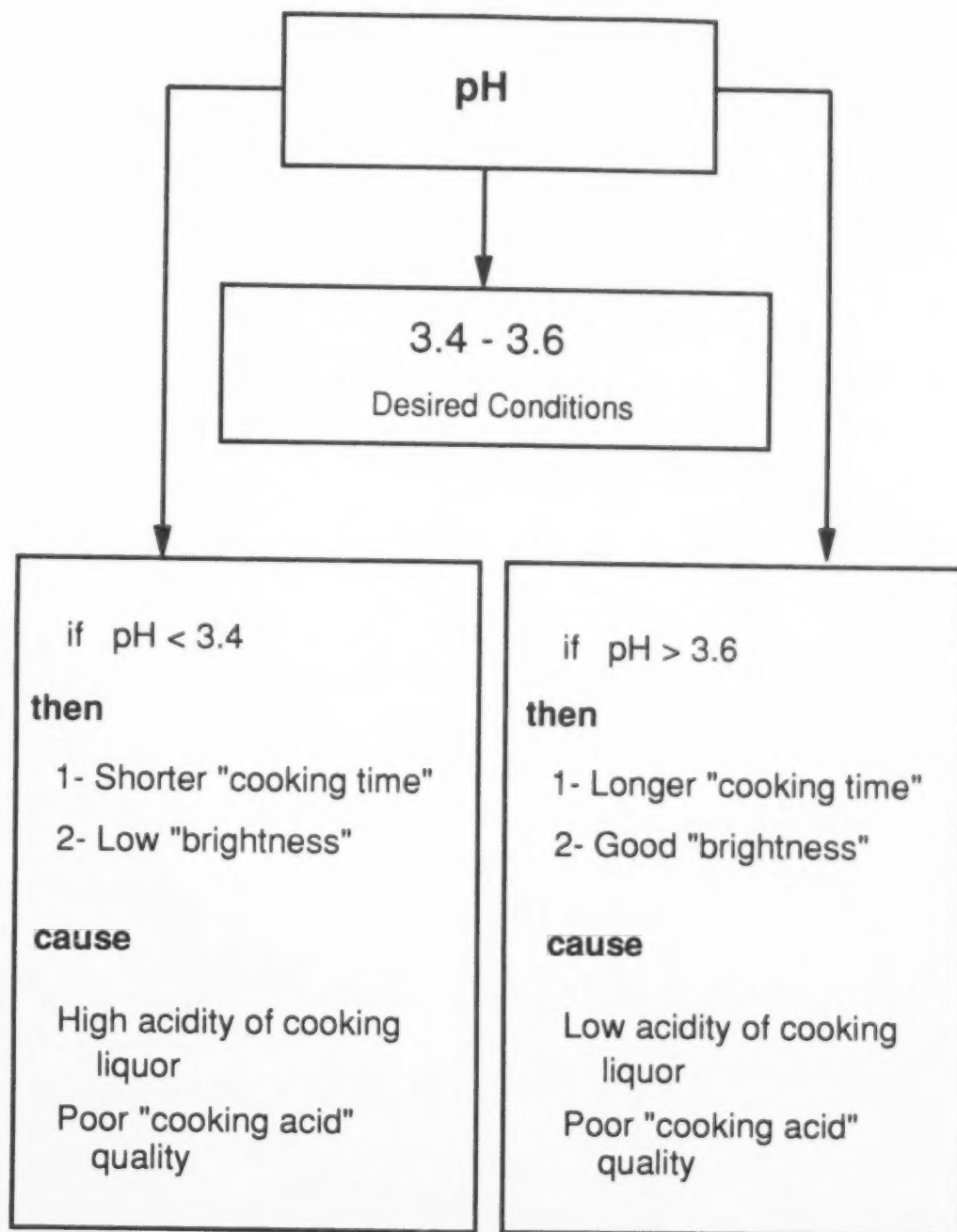


FIGURE 5.7 Qualitative knowledge for "pH" of cooking acid

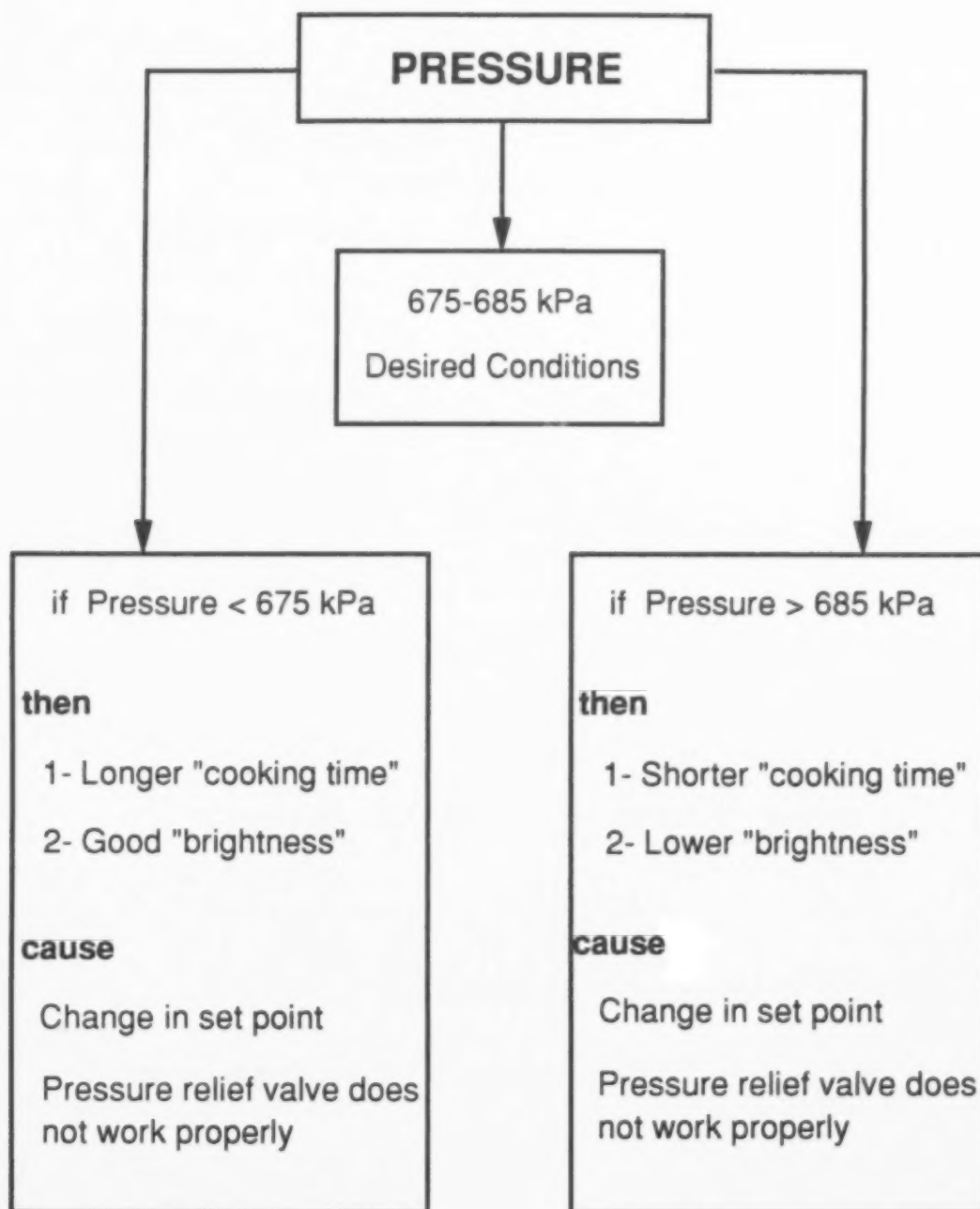


FIGURE 5.8 Qualitative knowledge for "Operating Pressure"

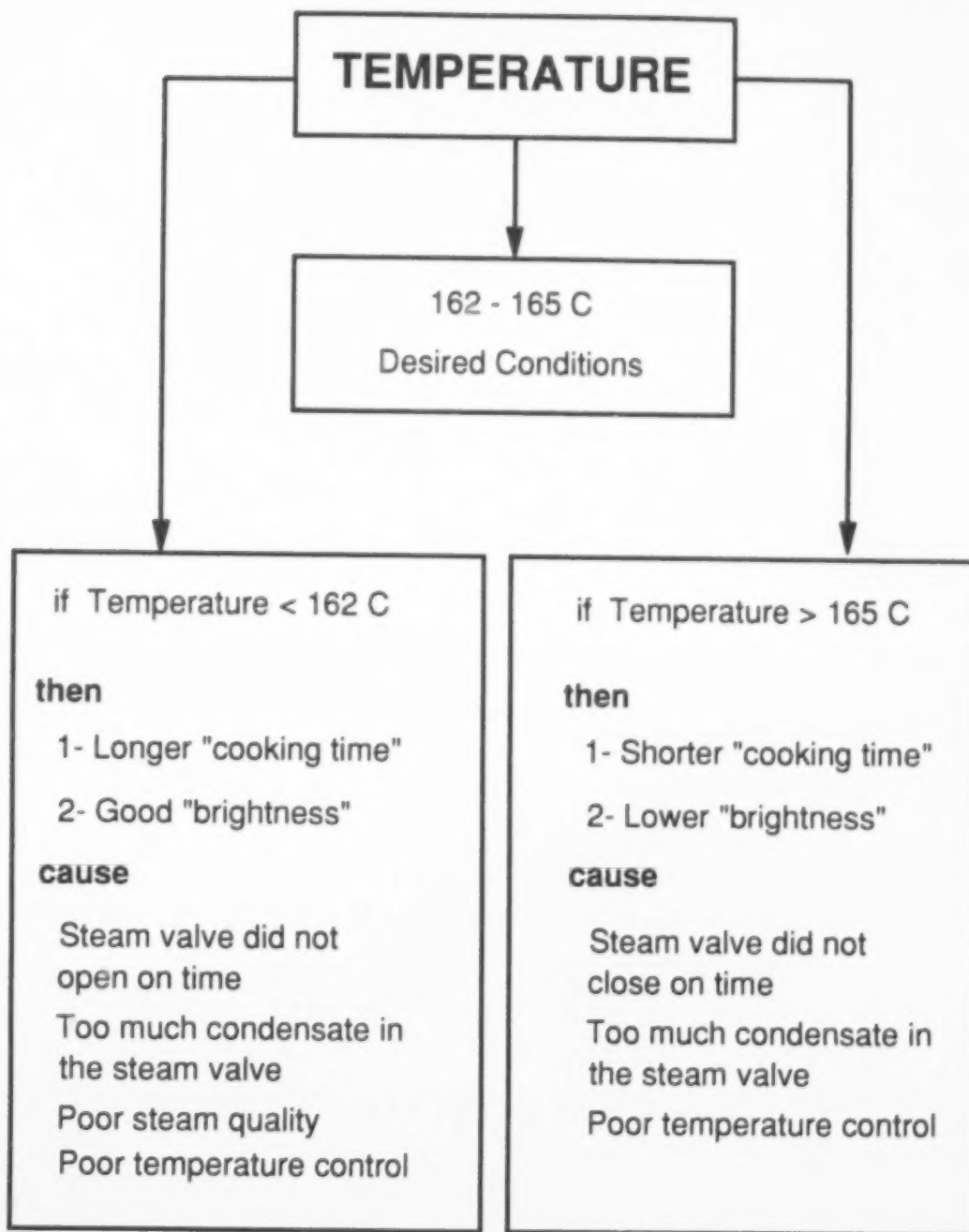


FIGURE 5.9 Qualitative knowledge for "Operating Temperature"

5.4 QUANTITATIVE KNOWLEDGE

This section describes a mathematical equation developed based on the "S-factor" model, which was previously discussed in Chapter 2. The model structure was first developed during a summer employment (summer 1986) for digester #1 [Haywood, 1989]. The model structure was used as a basis by the mill to develop a similar model for digester #4. As this IOSS project is based on digester #4, the second model is used as the quantitative knowledge for the IOSS [Levesque, 1989].

The model is an empirical mathematical equation which includes a modified S-factor. Only the total pressure measurement is available to the operators in the control room. Therefore, instead of using the partial pressure of SO_2 as employed by the "S-factor", the total pressure is used in a modified S-factor (i.e. S_m -factor)

As already mentioned in Chapter 2, the empirical mathematical model is represented by equation (5.1).

$$\text{Kappa No.} = a_1 - a_2 \cdot \int_0^t P_{\text{tot}} \cdot k \, dt + a_3 \cdot [\text{Total SO}_2 (0)] + a_4 \cdot [\text{Total SO}_2 (10)] \quad (5.1)$$

where	a_1, a_2, a_3, a_4	= Regression coefficients
	k	= Relative reaction rate
	P_{tot}	= Total pressure inside the digester, atm
	t	= time, hours
	Total $\text{SO}_2 (0)$	= Percentage of total SO_2 of the cooking liquor during acid filling (before cooking), %
	Total $\text{SO}_2 (10)$	= Percentage of total SO_2 of the circulating cooking liquor 10 minutes after the operating temperature has reached a maximum, %

The modified S-factor is represented by the following equation

The modified S-factor is represented by the following equation

$$S_m - \text{factor} = \int_0^t P_{\text{sa}} \cdot k \, dt \quad (5.2)$$

The relative reaction rate (k), can be calculated by the following equation

$$k = e^{\left[F - \frac{E}{R \cdot T}\right]} \quad (5.3)$$

where

E = activation energy, cal/mol

F = frequency factor, constant

R = gas constant, cal/mol-K

T = absolute cooking temperature, K

For sulphite pulping, the above parameters are replaced by the following numerical values

E = 20200 cal/mol

F = 27.25

R = 1.987 cal/mol-K

The numerical values of the regression coefficient in equation (5.1) are given as follows:

$a_1 = 108.2$

$a_2 = 0.02170$

$a_3 = 8.336$

$a_4 = 7.425$

The regression analysis based on the above parameters results in a coefficient of multiple correlation (R) of 0.958

Now, by setting the kappa number to 28 (i.e. target kappa number at Fraser's mill), equation (5.1) can be rearrange in the following manner to calculate the S_m -factor:

$$S_m - \text{factor} = \frac{28 - a_1 - a_3 \cdot [\text{Total SO}_2 (0)] - a_4 \cdot [\text{Total SO}_2 (10)]}{a_2} \quad (5.4)$$

The value for [Total SO₂ (0)] can be obtained during "acid filling" and the value for [Total SO₂ (10)] can be obtained from the circulating liquor during the cooking cycle (e.g. approximately 2:10 hour after the cooking cycle has started). Once both values are obtained, the target S_m-factor can be calculated using equation (5.4).

During the operation stage of "rising time", which consists of increasing the temperature and the total pressure to their operating conditions, the S_m-factor is calculated using equation (5.2), and is represented by S_{m1}-factor. However, when the temperature and the total pressure are at their operating conditions (i.e., does not changes with time), the S_{m2}-factor can be calculated by the following simplified equation as:

$$S_{m2} - \text{factor} = P_{\text{tot}} \cdot k \cdot \Delta t \quad (5.5)$$

By using both equation (5.2) for the "rising time" operation of the process and using equation (5.5) for the remaining part of the process, the total S_m-factor can be calculated.

$$S_m - \text{factor} = S_{m1} - \text{factor} + S_{m2} - \text{factor} \quad (5.6)$$

Equations (5.1) to (5.6) are used in the IOSS to predict the "cooking time", the "kappa number", and the "S_m-factor". The implementation of this model as well as its utilization are described in more details in Chapter 6.

CHAPTER 6

SYSTEM CONSTRUCTION

This chapter presents the construction and implementation of the "Intelligent Operation Support System" (IOSS). The structure of this system is organized in a similar fashion as shown in figure 3.3 in Chapter 3. IOSS is mainly implemented in C, while the numerical computation program is written in FORTRAN. Both the qualitative and quantitative knowledge are integrated together and coordinated by a meta-system [Rao, et al., 1989]. Such architecture facilitates the access of these two types of knowledge to the end user.

Chapter 6 begins with a section discussing the overall system structure. The first section gives a brief discussion on the directions engaged for the development of IOSS. The chapter then presents a detail description about the implementation of the database and the knowledge base of the system. A section discussing the phases of implementation, and the utilization of both qualitative reasoning system and the quantitative computation system is also included. This chapter ends with a section presenting the evaluation of IOSS.

6.1 OVERALL SYSTEM STRUCTURE

This section presents a discussion about the implementation platform and the programming languages utilized in the development of IOSS.

6.1.1 Implementation Platform

IOSS has been developed on an IBM-PC compatible computer (PC386/MSDOS). The objective of the project is to develop such a software program that it can directly be used in the industrial plant (digester's operating room). The main reason why a personal computer hardware is chosen is because the system has to be used in the Fraser plant. There are currently two IBM-PC compatible computers in the digester's operating room, and both can be used to install the IOSS. If the system were developed on a UNIX based workstation machine, the company would have to purchase such machine, and to train the operators how to use the UNIX based system. That is not possible at Fraser Inc. during the current bad economic situation.

6.1.2 Programming Languages

Majority of the IOSS are coded in C programming language, and developed using the Borland Turbo C++ compiler. The numerical computation program is written using FORTRAN. There are advantages and disadvantages in using conventional programming languages instead of using a commercialized expert system development shell.

The primary advantage for using conventional programming language is that it provides flexibility. IOSS is designed to support the digesters operation and to meet the operators' necessities. By using a conventional programming

language, the developer can customize the program, and modify it for the users' specific requirements. The interface of IOSS consists of a "main-menu", many "sub-menus", and different windows, which make the input/output of the information much easily interchanged between the user and the IOSS. A well developed I/O interface and "easy-to-use" computer program environment is very important for IOSS to be accepted by end users.

Another advantage for using conventional programming language is that once the system (IOSS) is completed and compiled, it can be install on any PC base computer without the purchase of any software or shell. Also, the code can be transferred to other operating platforms. If our system (IOSS) were developed using a commercialized shell, the company (Fraser Inc.) would have to purchase a license for the commercial package in order to install the system in its plant.

The main disadvantage of using conventional programming languages to build expert systems is that the development time is usually much longer than using a commercialized expert system shell. Many commercial packages provide built-in interface, and the structure of the knowledge base is already designed. Only the knowledge has to be entered into the system. This saves a lot of development time.

The following sub-section briefly describes all the function of the IOSS

6.1.3 Function Modules of IOSS

As already mentioned, the IOSS is a menu-driven system, which can be easily installed on any PC/MSDOS platform. The main function modules contain the following choices that can be selected by the user on the computer screen.

Main-menu #1

[Intro]	[Pulping]	[speC]	[Inputs]	[Situation]	[Option]	[Next menu]
---------	-----------	--------	----------	-------------	----------	-------------

Main-menu #2

[Prev. menu]	[Operation]	[Flow dia]	[Shutdown]	[startUp]	[Editor]
--------------	-------------	------------	------------	-----------	----------

When IOSS is running, the main-menu appears on the top of the display screen. Each selection from the main menu provides a sub-menu and/or a window system to display information to the user, or to receive information from the user. Some selection are linked together to process the information better. The main functions of each selection are described as follows;

[Intro]

This selection provides a brief introduction of the IOSS. It gives the user a brief description of each selection from main-menu #1 and main-menu #2.

[Pulping]

The "Pulping" selection gives an introduction on chemical sulphite pulping. It describes the purpose of digesting and the concept of delignification.

[Spec]

This sub-menu provides the specifications about equipment related to the chemical pulping process. It provides, for example, the type of pump with its capacity, or the size/material of a tank for a selected system. The IOSS provides information for the following equipment and system:

- Digesters
- Acid system
- Condensate system
- Water system
- Drain down system
- Dump system
- Flushing liquor system

[Inputs]

The "Inputs" module takes the data entered by the user into the IOSS, the data are held in a database. These data are the process quality inputs that describe a batch cooking cycle. These quality process inputs are presented in Table 5.1. The data required by the S_m -factor model are also entered from this sub-menu.

[Situation]

The "Situation" module provides the qualitative knowledge as well as the quantitative knowledge based on the status of the database (i.e. information entered through the "Inputs" selection). The qualitative knowledge includes estimated "cooking time" and remaining "cooking time", the effect of the quality inputs on the quality outputs, and the possible causes in which a certain quality inputs is out of its desired operation condition. The quantitative knowledge are the results from the S_m -factor model which provide on-line computed "cooking time", remaining "cooking time", and estimated "kappa numbers" during a cooking cycle.

If no information is entered into the database by the user prior the selection of [Situation], IOSS will utilize the default values from the database.

[Option]

This selection has similar function as the previous one (i.e. "Situation"). The "Option" selection can provide "cooking time" by using a bar graph. It also recommends "cooking time" of coupled quality inputs. This module can also display the desired conditions or the current conditions of all the process quality inputs in one single window.

[Next menu]

By choosing the "Next menu" selection, the top bar menu will be changed from main-menu #1 to main-menu #2.

[Prev. menu]

By choosing the "Prev. menu" selection, the top bar menu will be changed from main-menu #2 to main-menu #1.

[Operation]

This module provides all the operations to be performed for a single batch cooking cycle. The different operations provided by the system are basically the same ones described in Chapter 4.

[Flow dia]

This selection provides a graphical display of the batch digester system which includes the main piping, the circulating pump, the dump valve, and the liquor heater. The system also provides the initial cooking liquor quality, the initial chip quality, and the direction of the circulating liquor.

[Shutdown]

This selection provides the required steps on how to shutdown the operation of the digesters.

[Start-up]

This selection provides the required steps on how to start-up the operation of the digesters.

[Editor]

This "Editor" can access the user. The editor used in IOSS was "PE2", which is an IBM personal computer DOS editor. The main reason why an editor

is integrated into IOSS is the need to directly edit the files which are required for the proper operation of the IOSS. These files contain information about pulping, process specifications, process operations, and shutdown/start-up procedures.

6.2 DATABASE

The database of the IOSS is defined as an array containing the data of each process quality inputs. These values represent the status of a batch cooking process. The database array is given as:

float database[10]

As the configuration of IOSS is separated into several different C code modules, the database is defined as a global array so that more modules can use it. The specifier **float** means that the data in the database are held as "real" type data. Table 6.1 shows that all the process quality inputs are contained in the database.

Table 6.1: The set-up of database array

database[0]	= 38;	/*	CHIP LOAD	*/
database[1]	= 1.0;	/*	CHIP QUALITY	*/
database[2]	= 1.0;	/*	CHIP SIZE	*/
database[3]	= 5.75;	/*	TOTAL SO2 (0)	*/
database[4]	= 3.50;	/*	pH	*/
database[5]	= 163;	/*	TEMPERATURE	*/
database[6]	= 680;	/*	PRESSURE	*/
database[7]	= 2.80;	/*	COMBINE SO2	*/
database[8]	= 2.95;	/*	FREE SO2	*/
database[9]	= 0.00;	/*	TOTAL SO2 (10)	*/

All the values in Table 6.1 are default values. These values are automatically entered into the database when the IOSS starts running. However, each individual input can be modified by the user, who selects [Inputs] from main-menu #1. Each individual input can be selected from the sub-menu and its new values can be entered from the keyboard. All values in the database array except "database [9]" are used for processing the qualitative knowledge. Three values (temperature, pressure, total SO₂ (0), and total SO₂ (10) are used for processing the quantitative knowledge.

There is another array which contains the information about process behavior, and is defined as follow:

```
float s_fact[4]
```

This array is also a part of the database, and contains other required data for the quantitative computation system (i.e. S_m-factor model). Table 6.2 shows the information contained in the s_fact array.

Table 6.2: The set-up of s_fact array

s_fact[0]	/* S _m -factor (target) */
s_fact[1]	/* S _m -factor (10 min after operating temperature has been reached) */
s_fact[2]	/* The time when the cooking liquor sample for total SO ₂ (10) was taken, (hr) */
s_fact[3]	/* Estimated total cooking time */

The S_m-factor (target variable) is calculated by the mathematical model based on a kappa number of 28 (target quality criterion), and on the input data from the database array. The value for s_fact[1] is entered by the user and

consists of the S_m -factor at the time when the sample for the total % SO_2 (10) is collected. This S_m -factor (10) can be obtained from another computer system which is currently physically connected on digester #4 as a data acquisition system. Such system can calculate on-line S_m -factors. The `s_fact[2]` contains the time when the sample for total SO_2 (10) is taken. With the use of the first three numerical data, a "cooking time" can be calculated. The calculated "cooking time" is derived from the S_m -factor model and held in `s_fact[3]`.

Also from the information contained in the `s_fact` array, remaining "cooking times" and estimated kappa number can be calculated and provided to the operator for supervisory control. The calculation of all these variables will be discussed in more detail latter in this chapter.

6.3 KNOWLEDGE BASE ORGANIZATION

This section gives a description of the knowledge base organization for the qualitative reasoning system. We first present the overall structure of the qualitative knowledge representation. Then, a description about the reasoning process is presented. Finally, the special features that are implemented into IOSS to make the system more efficient are included in the last sub-section.

6.3.1 Overall Structure

The qualitative knowledge for each quality input variables are held in different arrays and matrices. Each array or matrix contains different type of quality information, and is kept separately. The knowledge contained in these arrays and matrices are describes below.

```
float ranges[][2]
```

By defining a matrix such as "float ranges[][2]" means that its dimension is [x] by [2], where x can be any positive integer. Therefore, ranges[][2] has no specified dimension in the x direction, where the memory of the computer is the limit.

The data contained in this matrix are the different possible values of each process input variable that IOSS can process. Most process input variables are separated in a maximum of 5 different quality ranges. The values contained in ranges[][0] are the low limit of each range, and the values contained in ranges[][1] are the upper limit of each range. The reason why the range matrix has an unspecified dimension is to add more situations if the system needs to be modified. The current system can manage up to 10 different values for each process input variable.

```
char *status[10]
```

The "status" array contains quality information about the input variables. Each input variable has its own status "array". The quality information contained in the status "array" is, for example, whether the value of the input variable is "high", "low", or "desired".

```
char *time_required[10]
```

The "time_required" array contains the estimated required "cooking time". Each value in the "ranges" matrix has a corresponding "cooking time".

```
char *situation[10]
```

All the other type of quality information about different values of each variable are held in an array called "situation". The information contained in this array are, for example, the results of the final pulp quality (e.g. "high", "low", or

"good" pulp brightness) due to a certain quality of the input variable, as well as the possible causes why the value of the input variable is out of the desired quality range. In some cases, the suggestions (e.g. what to do) are stored quality range ("situation" array).

It should be mentioned that at this point, the information contained in the above arrays and matrices are displayed to the user through a window system.

```
char *time_limits[][2]
```

As IOSS provides the user some graphical displays (e.g. bar graph) to show the effect of each input variable on the "cooking time", an array "time_limit" is used to hold the high limit and low limit of an estimated range for "cooking time". For example, if `time_limits[][0]="+20"`, and `time_limits[][1]="+10"`, then the bar on the "bar graph" for that particular input variable will show a yellow colored section indicating that the estimated "cooking time" is about 10 to 20 minutes more than the normal required "cooking time".

```
float weight[10]
```

The "weight" array contains weight numerical values for the input variables. These weight values are used for the "bar graph" to couple the effect from several input variables on the "cooking time". For a case where less "cooking time" is required for the cooking cycle to achieved the desired pulp quality, the corresponding weight value in the "weight" array would contain a negative (-ve) value. When more cooking time is required, the weight will be a positive (+ve) value. Under desired condition, the weight value is equal to zero.

Example 6.1 shows a sample of the knowledge representation for one single input variable. The input variable in this example is total SO₂ of the initial cooking acid. The knowledge for other input variables are represented in the same way as Example 6.1.

EXAMPLE 6.1: Knowledge base organization for total SO₂

```

/* ***** */
/* **** TOTAL SO2 OF ACID **** */
/* ***** */

int total_range(float total)
{
    int loc;

    /* Definition of Ranges of "total SO2" for each situation */

    /* TOTAL SO2 RANGES SETUP */
    /* -----*/

    float ranges[][2]=
    {4.00,4.80, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
     4.81,5.44, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
     5.45,6.15, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
     6.16,6.50, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
     6.51,7.40, // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
     0.00,0.00, // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
     0.00,0.00}; // Situation #6 (LOW_limit[][0],HIGH_limit[][1])

    loc = rule_base_setA(total,ranges);
    return loc;
}

char *total_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* TOTAL SO2 STATUS SETUP */
    /* -----*/

    status[0]="LOW TOTAL SO2 OF ACID\n"
              "-----";

    status[1]="LOW TOTAL SO2 OF ACID\n"
              "-----";

    status[2]="NORMAL CONDITIONS\n"

```



```

"      1- Good Pulp Brightness      \n"
"                                     \n"
" COOKING ACID QUALITY               \n"
"                                     \n"
"      1- POOR Quality               \n"
" ";

situation[1]=
"      UNDER THIS CONDITION          \n"
"      -----                        \n"
"                                     \n"
" RESULTS IN THE FINAL PULP:         \n"
"                                     \n"
"      1- Good Pulp Brightness       \n"
"                                     \n"
" COOKING ACID QUALITY               \n"
"                                     \n"
"      1- POOR Quality               \n"
" ";

situation[2]=
"      GOOD % TOTAL SO2 IN ACID      \n"
"      -----                        \n"
"                                     \n"
" RESULTS IN THE FINAL PULP:         \n"
"                                     \n"
"      1- Good Pulp Brightness       \n"
"                                     \n"
"      2- Good Kappa Number          \n"
"                                     \n"
" ";

situation[3]=
"      UNDER THIS CONDITION          \n"
"      -----                        \n"
"                                     \n"
" RESULTS IN THE FINAL PULP:         \n"
"                                     \n"
"      1- Pulp Brightness could be lower \n"
"      (check situation of FREE & COMBINE SO2) \n"
"                                     \n"
" COOKING ACID QUALITY               \n"
"                                     \n"
"      1- Strong Cooking Acid        \n"
" ";

situation[4]=
"      UNDER THIS CONDITION          \n"
"      -----                        \n"
"                                     \n"
" RESULTS IN THE FINAL PULP:         \n"
"                                     \n"
"      1- Pulp Brightness could be lower \n"
"      (check situation of FREE & COMBINE SO2) \n"
"                                     \n"
" COOKING ACID QUALITY               \n"
"                                     \n"
"      1- Strong Cooking Acid        \n"
" ";

situation[5]=" ";
situation[6]=" ";

```

```

return situation[loc];

}

char *total_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* TOTAL SO2 TIME_LIMITS SETUP */
    /* -----*/

    // the pH high_time[0] and low_time[1] limits

    time_limits[0][0]="+20";
    time_limits[0][1]="+10";

    time_limits[1][0]="+10";
    time_limits[1][1]="+0";

    time_limits[2][0]="GOOD";
    time_limits[2][1]="GOOD";

    time_limits[3][0]="-0";
    time_limits[3][1]="-10";

    time_limits[4][0]="-10";
    time_limits[4][1]="-20";

    time_limits[5][0]="GOOD";
    time_limits[5][1]="GOOD";

    time_limits[6][0]="GOOD";
    time_limits[6][1]="GOOD";

    return time_limits[loc][hl];
}

float total_weight(int loc)
{
    float weight[10];

    /* TOTAL SO2 WEIGHT SETUP */
    /* -----*/

    weight[0]=3.0;
    weight[1]=1.0;
    weight[2]=0.0;
    weight[3]=-1.0;
    weight[4]=-3.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

```

The knowledge base of the qualitative reasoning system is coded in a separate module called "KNBASE.C". If the knowledge needs to be updated or modified, the developer can add, delete, or modify the content of the arrays and matrices. The knowledge base can be easily modified without changing the logic or the rule structure. Even people who does not have background in "C" language programing can accomplish the modification of knowledge base in IOSS.

6.3.2 Qualitative Reasoning System

The reasoning process of the qualitative system is exhibited in Figure 6.1. The quality data about the process (i.e. Digester) are entered into the database by the user. These data are processed through the knowledge base by the inference engine. The knowledge base which consists of arrays and matrices, receives the data in the module containing the "ranges" matrix. The "ranges" matrix and the values of that particular input are processed through a "variable rule" system to find the location (loc) in the "ranges" matrix that satisfies the condition. The location (loc) is returned to the inference engine. The inference engine feeds the location (loc) back into the knowledge base for further processing by the remaining arrays (i.e. "status", "time_required", and "situation"). The information in these arrays at the location (loc), which contains the qualitative knowledge corresponding to the value of the input variable being processed, can be displayed on the video screen of the computer. The information provided by the quantitative reasoning system can be used by the human operators for on-line supervisor control.

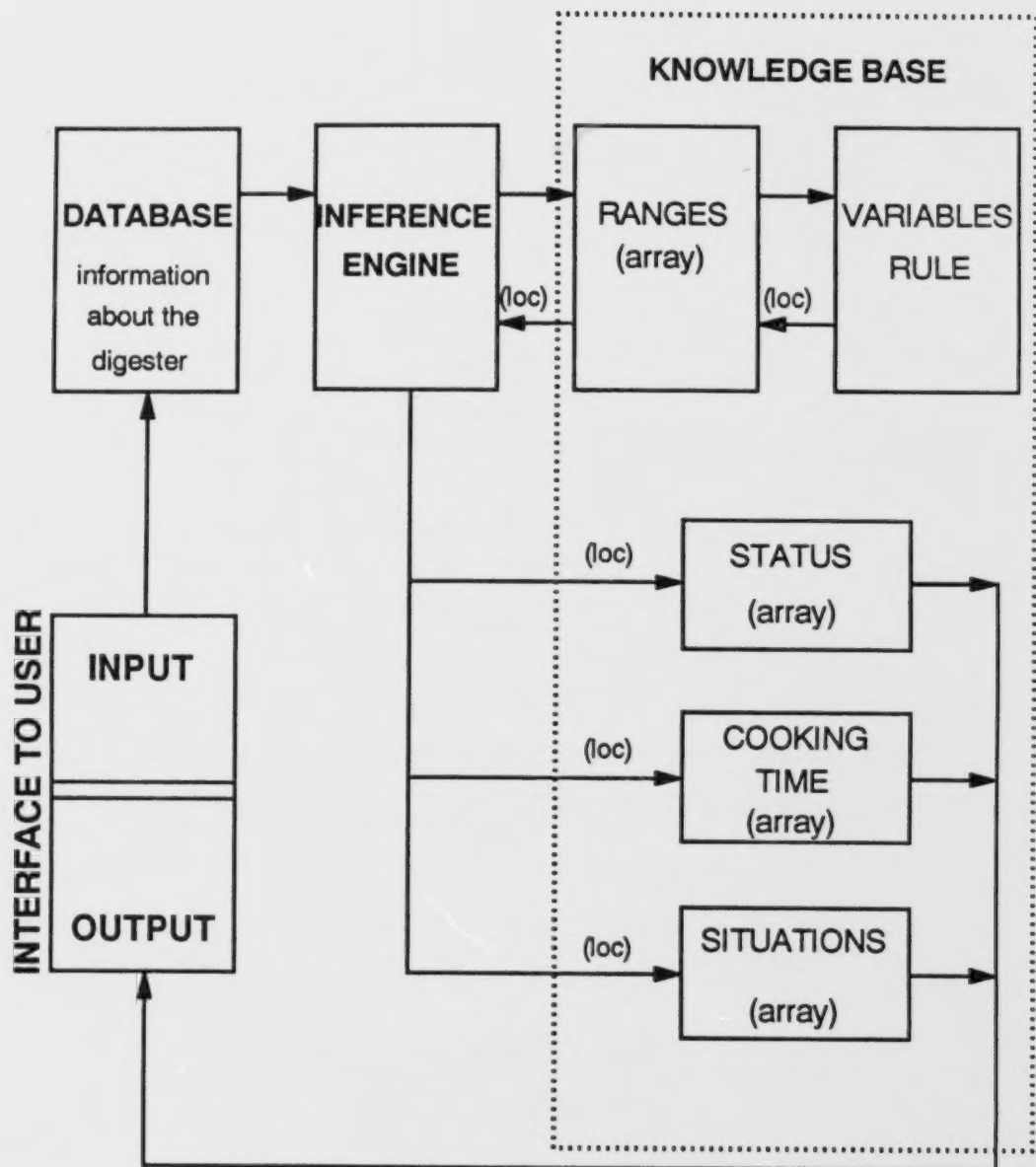


FIGURE 6.1 Reasoning process of "qualitative system"

6.3.3 Advanced System Features

This sub-section presents the different features employed by iOSS to better process the knowledge.

Coupling System

In the cases where more input variables are not within their desired value, the system needs to couple the effect on the cooking behaviors and derive a coupled required "cooking time". Figure 6.2 shows how the input variables can be coupled together. As demonstrated in Figure 6.2, different "cooking times" can be obtained at three different levels. Level I derives "cooking time" for each individual input variable. Level II provides coupled "cooking time" for the "Wood Chips" (the system couples the effect of "chip size", "chip load", and "chip quality" together), the "Cooking Acid" (the system couples the effect of the "total SO_2 " and "pH" together), and the "Operating Condition" (the system couples the effect of the "Operating Temperature" and the "Operating Pressure" together). The Level III of the coupled system provides "cooking time" for all the input variables together. The coupled "cooking time" is provided to the user through the bar graph display under the [Options] selection in main-menu #1.

The compilation of the effect of these input variables on the "cooking time" is done by adding the weight values in the "weight" array of each input variables. Table 6.3 gives the weighted values for the different possible values of all input variables. Table 6.4 provides the required "cooking time" corresponding to different weight values. If a cumulated weight value is higher or lower than those presented in the tables, the system will not be able to provide the coupled "cooking time". Table 6.5 shows the number of possibilities that iOSS can handle

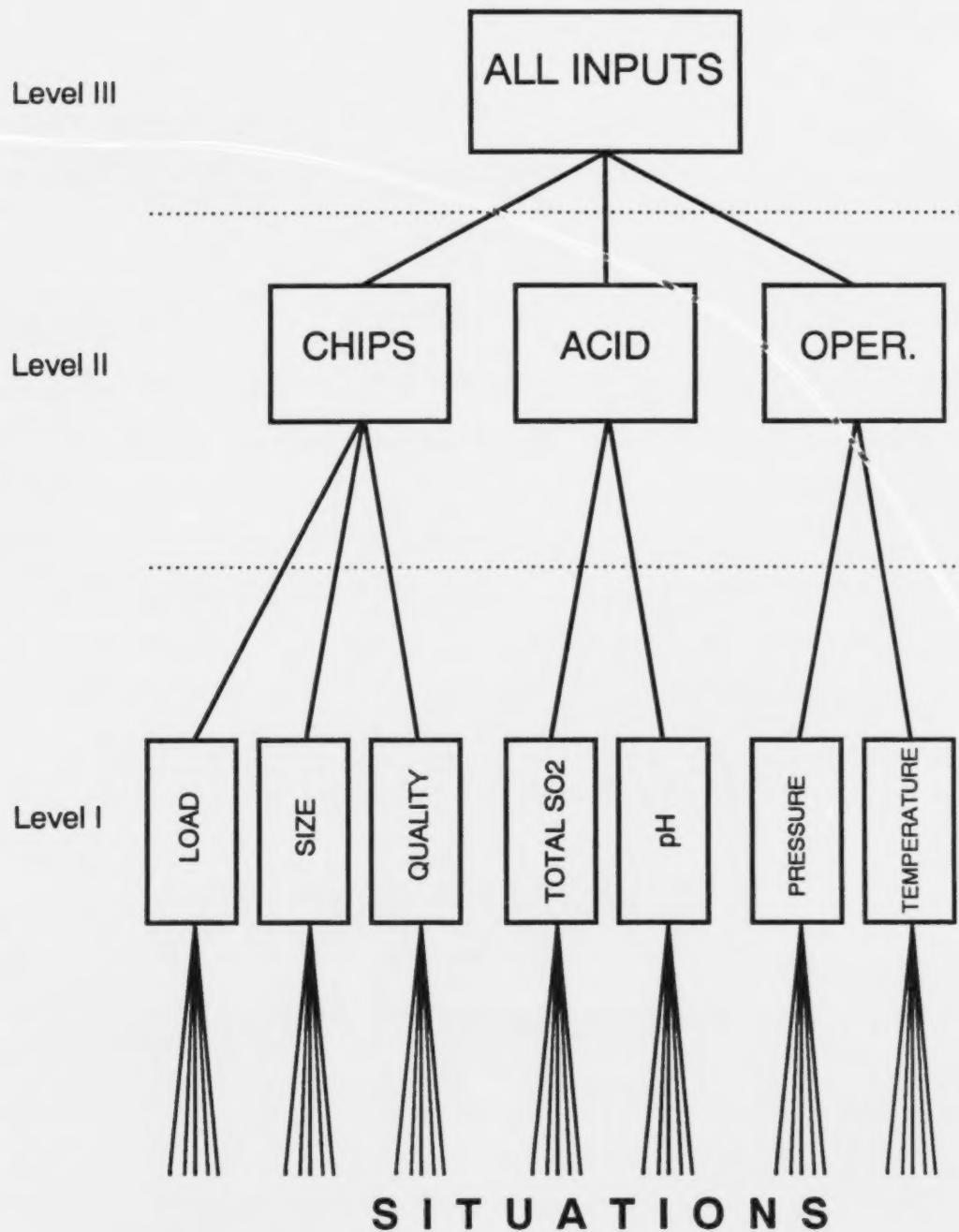


FIGURE 6.2 Structure of Knowledge Base

at each levels. A total of 37500 different possible combinations exist at level III of the coupled system.

TABLE 6.3: Description of each situation at level I

Individual Inputs Quality	Description of Each Situation (RULE)	Weight Values
Wood Chips Load	22 % gauge to 29 % gauge	-4
	30 % gauge to 34 % gauge	-2
	35 % gauge to 40 % gauge	0
	41 % gauge to 46 % gauge	+2
	47 % gauge to 55 % gauge	+4
Wood Chips Quality	Good quality	0
	Old chips	-3
	More than 10% hardwood	-3
	"Old chips" & "more than 10% hardwood"	-30
Wood Chips Size	Standard Size	0
	Over Standard Size	+3
	Under Standard Size	-3
Total % SO ₂ of Initial Cooking Liquor	4.00 % to 4.80 %	+3
	4.81 % to 5.44 %	+1
	5.45 % to 6.15 %	0
	6.16 % to 6.50 %	-1
	6.51 % to 7.40 %	-3
pH of Initial Cooking Liquor	2.5 to 2.9	-3
	3.0 to 3.3	-1
	3.4 to 3.6	0
	3.7 to 4.0	+1
	4.1 to 4.6	+3
Operating Temperature	154 °C to 157 °C	+4
	158 °C to 161 °C	+2
	162 °C to 165 °C	0
	166 °C to 169 °C	-2
	170 °C to 175 °C	-4
Operating Pressure	575 kPa to 624 kPa	+3
	625 kPa to 674 kPa	+1
	675 kPa to 685 kPa	0
	686 kPa to 735 kPa	-1
	736 kPa to 785 kPa	-3

TABLE 6.4: Cooking time ranges for the coupled weight values (span of 10 minutes)

Weight Values	Impact on Cooking Time	Corresponding Required Cooking Time
+9	40 - 50 min longer	4:35 - 4:45 hr
+8	35 - 45 min longer	4:30 - 4:40 hr
+7	30 - 40 min longer	4:25 - 4:35 hr
+6	25 - 35 min longer	4:20 - 4:30 hr
+5	20 - 30 min longer	4:15 - 4:25 hr
+4	15 - 25 min longer	4:10 - 4:20 hr
+3	10 - 20 min longer	4:05 - 4:15 hr
+2	05 - 15 min longer	4:00 - 4:10 hr
+1	00 - 10 min longer	3:55 - 4:05 hr
0	Normal Cooking time	3:45 - 3:55 hr
-1	00 - 10 min shorter	3:35 - 3:45 hr
-2	05- 15 min shorter	3:30 - 3:40 hr
-3	10 - 20 min shorter	3:25 - 3:35 hr
-4	15 - 25 min shorter	3:20 - 3:30 hr
-5	20 - 30 min shorter	3:15 - 3:25 hr
-6	25 - 35 min shorter	3:10 - 3:20 hr
-7	30 - 40 min shorter	3:05 - 3:15 hr
-8	35 - 45 min shorter	3:00 - 3:10 hr
-9	40 - 50 min shorter	2:55 - 3:05 hr

TABLE 6.5: Number of all possible situations

Individual quality inputs (LEVEL I)	# Possible Situations	Coupled quality inputs (LEVEL II)	# Possible Situations	Coupled quality inputs (LEVEL III)	# Possible Situations
Chip Load	5	WOOD CHIPS	5x4x3 = 60	ALL QUALITY INPUTS	60x25x25 = 37,500
Chip Quality	4				
Chip Size	3				
Total % SO2	5	COOKING LIQUOR	5x5 = 25		
pH	5				
Operating Temperature	5	OPERATING CONDITIONS	5x5 = 25		
Operating Pressure	5				

Common Sense Reasoning Mechanism

Generally speaking, a computer system will take any input a user provides, even wrong input data. In real industrial application, it is very dangerous. The data to be entered into the IOSS which describes the process status must be physically significant. In order to prevent a user from inputting wrong data to IOSS, a simple common sense reasoning mechanism is built into the interface of IOSS. For example, if the user inputs the operating temperature at a value of 200 °C, IOSS will not accept this value because the value is physically impossible (i.e. out of working range). When this happens, IOSS automatically requests another value. If a new value is not entered, the corresponding input variable will get the default value which is the desired value. The rules which perform the screening of the non-physical numerical values are executed at the interface level so that these non-physical wrong data will not be used in the knowledge base, thus it makes the system run more efficiently.

Variable Rules

The level I of the knowledge is designed so that only 2 rules are utilized. This new technique is called "variable rule". Such technique is used to better organize the knowledge base as well as to improve the knowledge processing and the running time.

As the knowledge base is built with arrays and matrices, the "variable rule" system is used to find the location (loc) in a matrix that satisfies a certain value of an input variable. The syntax of variable rules is shown in Example 6.2. The first variable rule structure is defined as "rule_base_setA", and its structure is in line 6 of Example 6.2. This structure (rule_base_setA) is used to process input variables whose value are numerical data (i.e. chip load, total SO₂ , pH, operating temperature, and operating pressure)

Example 6.2: Syntax of the variable rules

```
int rule_base_setA(float xx, float rangel[][2])
{
    int i;

    for(i=0; i<=9; i++){
(line 6)    if (xx >= rangel[i][0] && xx <= rangel[i][1])
        break;
    }
    return i;
}

int rule_base_setB(float xx, float par[])
{
    int i;

    for(i=0; i<=9; i++){
(line 17)    if(xx == par[i])
        break;
    }
    return i;
}
```

Another variable rule, which is defined as "rule_base_setB", is used to process the remaining input variables, which are "chip quality" and "chip size". The values of these input variables are defined with "words" rather than numerical values. The structure of this variable rule is in line 17 of Example 6.2. The variable rule system is coded as a separate module called "RULEBASE.C".

6.4 QUANTITATIVE COMPUTATION

As already discussed in previous chapters, IOSS has a quantitative computation system to calculate "cooking times" and "kappa numbers". The mathematical model implemented in IOSS is based on the S_m -factor and presented by Equation 5.4. This system is programmed with FORTRAN language and its module is called "COOKMODL.FOR".

The following sub-sections describe the computation process to calculate the S_m -factor, and the equation used to calculate "cooking time" and "kappa number".

6.4.1 Computation Process

The computation process of the quantitative system is shown in Figure 6.3. The required data from the "database" array and the "s_fact" array (i.e. from the DATABASE) are passed to the numerical computation module by the inference engine to calculate the target S_m -factor. Here, the target S_m -factor is calculated based on the target kappa number (i.e. 28). Total SO_2 (10) is required for the computation of S_m -factor, which can only be obtained 10 minutes after the operating temperature has reached the maximum value. Usually, it takes approximately 2 hours for the temperature to reach a maximum, that is,

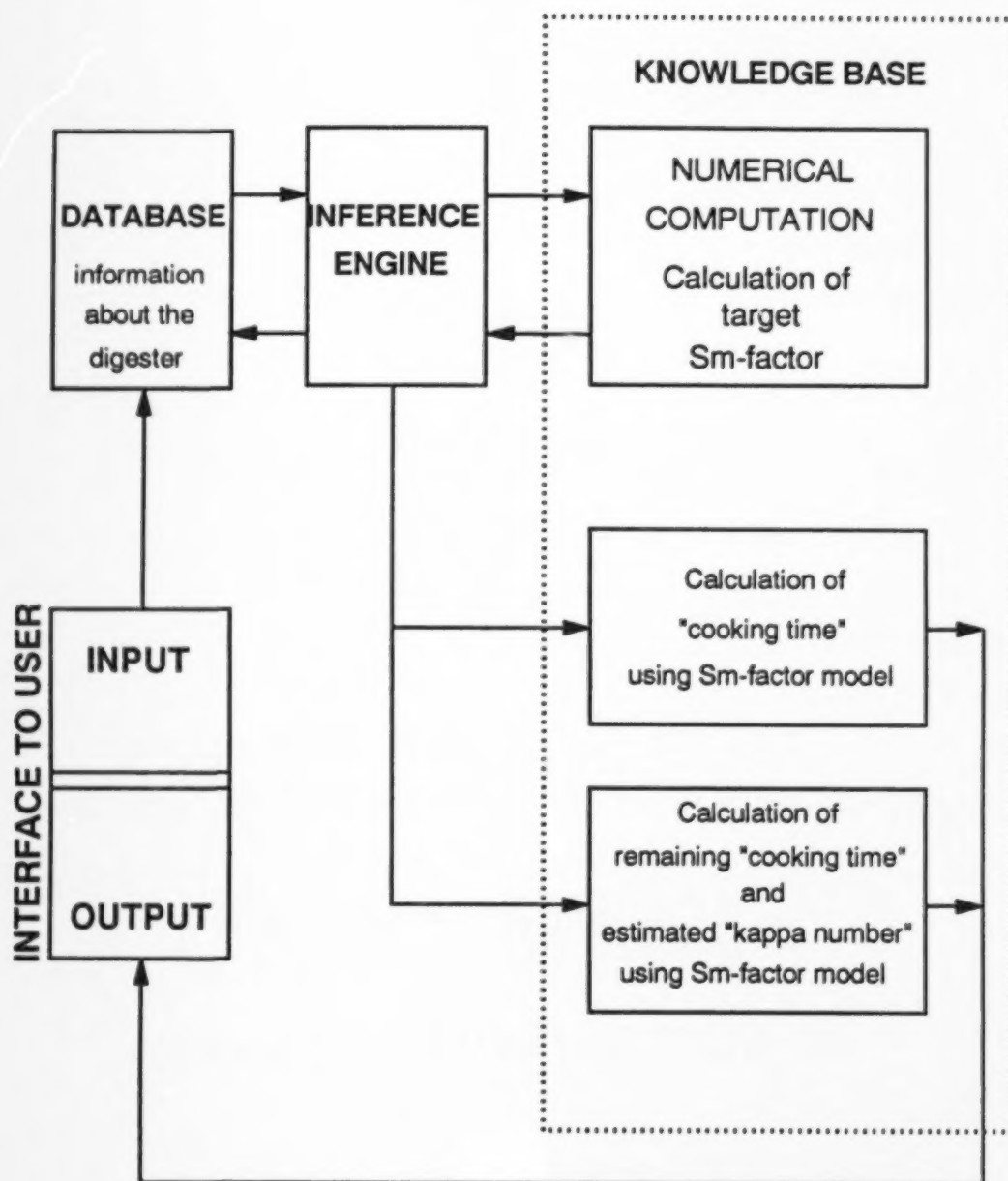


FIGURE 6.3 Computation process of "quantitative system"

the target S_m -factor can only be calculated approximately 2:10 hours after the cooking cycle has been started.

Once a target S_m -factor is calculated, the inference engine passes the calculated S_m -factor into different modules to calculate "cooking times" and "kappa numbers". These results are then displayed on the computer screen through the user interface.

6.4.2 Calculation of "Cooking Time"

As discussed previously in Chapter 5, in order to calculate a "cooking time" the system requires the S_{m1} -factor and the time when the cooking liquor sample for the total SO_2 (10) test is taken. S_{m1} -factor is computed at different time interval. Once the temperature and pressure of the digester are held constant at their operating conditions, the S_{m2} -factor can be calculated using Equation (5.5). Therefore, by combining Equations (5.5) and (5.6) together, the required "cooking time" for the production of good quality pulp can be calculated. The following equation can demonstrate this calculation:

$$\text{"cooking time"} = \left[\frac{S_m - \text{factor}(\text{target}) - S_{m1} - \text{factor}}{P_{\text{tot}} \cdot k} \right] + t_1 \quad (6.1)$$

where

t_1	=	The time that the total SO_2 (10) was taken, hour
P_{tot}	=	Operating pressure, atm
k	=	Relative Reaction rate

6.4.3 Estimation of "Kappa Number"

When the process reaches its operating temperature and pressure, the user can obtain an estimated current pulp kappa number. The kappa number can be calculated using Equation (5.1) based on the value of the S_{mt} -factor at the requested time. S_{mt} -factor can be calculated using the following equation:

$$S_{mt} - \text{factor} = S_{mi} - \text{factor} + P_{tot} \cdot k \cdot [t - t_1] \quad (6.2)$$

where

t = Any particular time after the target S_m -factor is calculated

The remaining "cooking time" can also be calculated from the following equation:

$$\text{remaining "cooking time"} = \text{"cooking time"} - t \quad (6.3)$$

All results produced by the quantitative computation system can be used by the operators for on-line supervisor control.

6.5 PHASES OF IMPLEMENTATION

Chapter 4 discussed the different phases for implementing intelligent systems to industrial process applications. IOSS is implemented at phases 1 and 2 of the implementation procedure where the data are entered by the user using a key board. Currently, IOSS is an off-line system that is not physically connected to the instrumentations of the digesters. However, IOSS can provide on-line supervisory control which can help the operators controlling the digesters' operation.

Both qualitative and quantitative systems can be used by the operators for decision-making. However, as the mathematical model is only a function of the "pressure", "temperature", "total SO_2 ", and "time", the quantitative computation system cannot be used for situations when the other input variables (i.e. "chip load", "chip load", "chip quality", and pH of acid) are out of their normal quality conditions. Also, the quantitative computation system does not provide qualitative information about the effect of input variables (such as effect on pulp brightness, pulp strength, etc) on the final pulp product.

As the numerical computation generates the rigid solutions, the quantitative computation system should be used first to predict required "cooking times" for the production of good quality pulp, when the mathematical model is applicable.

The qualitative reasoning system can be used for any situation by the end users to obtain quality information about the process behavior. The qualitative system does not guarantee perfect solutions, but it can give direction about the process behavior changes. Such system (i.e. quantitative reasoning system)

increases the interactions between the operators/users and the actual process (digester).

6.6 EVALUATION

This section discusses the evaluation of IOSS by different industrial experts and professional research scientists.

During the development of IOSS, Intelligence Engineering Laboratory, university of Alberta has organized numerous discussion and evaluation meetings for the project. Many important suggestions help the improvement of IOSS.

After IOSS had been developed and evaluated in our laboratory, the program back was sent to Fraser Inc. for field testing. Their technical control superintendent Mr. Gilles Volpe wrote to us to state that IOSS is "well developed expert system, and applicable into the operation for the operators and the process engineers. A lot of useful information is contained in this program with the possibility of expanding for the future Fraser's cooking model" [Volpe, 1991].

Mr. Andrew M. Slowik, the manager of process systems at Fraser, had also evaluated the program. After Mr. Slowik had used the system, a meeting and demonstration was organized with other staff members for further evaluation. Included at the meeting were the superintendent, technical assistants, and area engineers from the digester's operation. Attending were also representatives from Central Technical, Plant Engineering, and Maintenance departments. According to Mr. Slowik, the response for the system (IOSS) was extremely encouraging. All department representatives at the demonstration had positive comments on the IOSS's ease to use and quality of

the information gleaned from its use. Many of them thought that IOSS could be an excellent training tool for all trades and disciplines involved in the digester's operation, especially production personnel. Also mentioned in Mr. Slowik's letter, was that several people suggested that the system could be considered as a reference for engineering/maintenance due to the completeness of information contained within IOSS [Slowik, 1991].

The project has been presented to several pulp and paper industrial companies as well as research institutes, and was discussed with application domain experts and AI development experts to evaluate and modify IOSS. A list of the presentation and evaluation at those companies is given as follows:

1. Alberta Research Council, Computer Engineering Department, Calgary, Alberta, April 1991
2. Procter & Gamble Cellulose, Grand-Prairie, Alberta, May 1991
3. Millar Western Pulp Ltd., Edmonton, Alberta, July 1991
4. Ranger Pulp & Paper Inc., Edmonton, Alberta, September 1991
5. Alberta Newsprint Company, Whitecourt, Alberta, September 1991
6. Millar Western Pulp Ltd., Whitecourt, Alberta, September 1991

Based on these presentations, the industrial staff felt that such project and idea are a very good model for real industrial process applications.

CHAPTER 7

ILLUSTRATIONS

This chapter presents an illustration of the running process of IOSS. It gives the reader a better understanding how IOSS works. We first present an example of the qualitative reasoning system, then, an example of the quantitative computation system. All figures included in Chapter 7 are the hard copies directly from the computer screen.

7.1 QUALITATIVE REASONING

This section presents an example of executing the qualitative reasoning system which can be used by the operators in the digester's control room for operation support. Let us select a particular batch cycle where the input variables have the following data:

chip load	32% gauge
chip quality	good quality
chip size	normal size
free % SO ₂	2.79 %
combined % SO ₂	2.60 %
total % SO ₂	5.39 %
pH	3.4
operating temperature	160 C
operating pressure	680 kPa

The user can enter these values into IOSS by selecting the [inputs] sub-menu from main menu #1. Figure 7.1 is an example to explain how the value for total SO₂ of the initial cooking acid is entered into IOSS through a window input system. All of the other input variables can be entered in the same way.

Once all the required data are entered, the user may first select the [Situation] sub-menu from main-menu #1 to obtain the qualitative information about the process behavior based on the data stored in the database. Figures 7.2 to 7.4 show the qualitative information for "chip load", "total SO₂", and "operating temperature". Figure 7.5 demonstrates the qualitative information for the "operating pressure". As the values of the other input variables are within desired quality, they have no impact on the normal cooking cycle behavior.

After the user has received all the quantitative information, the user may select [Bar graph] in the [Option] sub-menu (Figure 7.6). The bar-graph system provides the estimated "cooking time" through a graphical style window. The first "bar graph" shows the estimated cooking time for each of the input variables (Figure 7.7). The normal range for the "cooking time" (i.e. 3:45 - 3:55 hours) is given by a light gray colored horizontal bar on the bar-graph window. The input variable whose vertical "bar" is within the normal "cooking time" range tells us that the quality is within desired value range. The input variable whose vertical "bar" in the bar graph is divided into two sections indicates that its quality is out of desired conditions. The top section of the vertical "bar" (in Figure 7.7 light gray color) for undesired quality range input variables indicates the lower and upper limits of the estimated "cooking time" range.

Upon pressing the "PgDn" key on the key board, the next "bar-graph" appears on the video screen, which provides the coupled "cooking time" for

"wood chips", "cooking acid", and "temp-press" (Figure 7.8). Figure 7.9 is the page 3 of the bar-graph system, and it shows an estimated "cooking time" by coupling all the process input variables together.

7.2 QUANTITATIVE COMPUTATION

This section presents an example of running the quantitative computation system. The example used to illustrate the quantitative computation system is based on the following set of data for a batch cooking cycle:

chip load	38% gauge
chip quality	good quality
chip size	normal size
free % SO ₂	3.05 %
combined % SO ₂	2.90 %
total % SO ₂	5.95 %
pH	3.5
operating temperature	165 C
operating pressure	680 kPa

All those data are entered into IOSS in the same way as explained in the previous example. To execute the quantitative computation system, the user may select [s-factor model] from the sub-menu [Inputs] as shown in Figure 7.10. Upon selecting the "S-factor model", IOSS will ask the user to enter the other required data for the mathematical model (Figure 7.11).

total % SO ₂ (10)	1.89 %
TIME of sample (t ₁)	2:10 hours
S _{m1} -factor	125 atm-hour

Once all required data are entered into IOSS, a target S_m -factor is calculated (Figure 7.12). Based on the target S_m -factor calculated, the required "cooking time", the estimated "kappa number", and the remaining "cooking time" can be calculated.

To obtain a calculated "cooking time" for the particular batch cook, the user may select [Cooking time] from the [Situation] sub-menu. The required "cooking time" for this cooking cycle is 3:41 hours (Figure 7.13).

The remaining "cooking time" and estimated "kappa number" can be obtained by selecting [Time-Kappa] from the [Situation] sub-menu. Then, the current cooking time of digesting batch cycle (i.e. how much time the chips have been cooking) is entered into IOSS. Finally, the "remaining cooking time" and estimated "kappa number" are obtained. Figure 7.14 shows the results as the "remaining cooking time" is "0:18 hours", and the estimated "kappa number" is 31. This feature aims at providing supervisory control function.

The total required "cooking time" calculated using the S_m -factor mathematical model can also be displayed in the bar-graph system, as operators usually prefer looking at results in a graphical representation form (Figure 7.15).

inTro	Pulping	speC	Inputs	Situation	Option	Next menu
SO₂ TOTAL What is the total SO₂ of cooking acid in percent during acid filling ? ENTER TOTAL SO₂ --> 5.39 Percent press ENTER to continue						
				ity 02 SO₂ 2 Mperature essure model ef		
"INTELLIGENT OPERATION SUPPORT SYSTEM" FOR THE BATCH DIGESTERS						

FIGURE 7.1 Input data for "total SO₂"

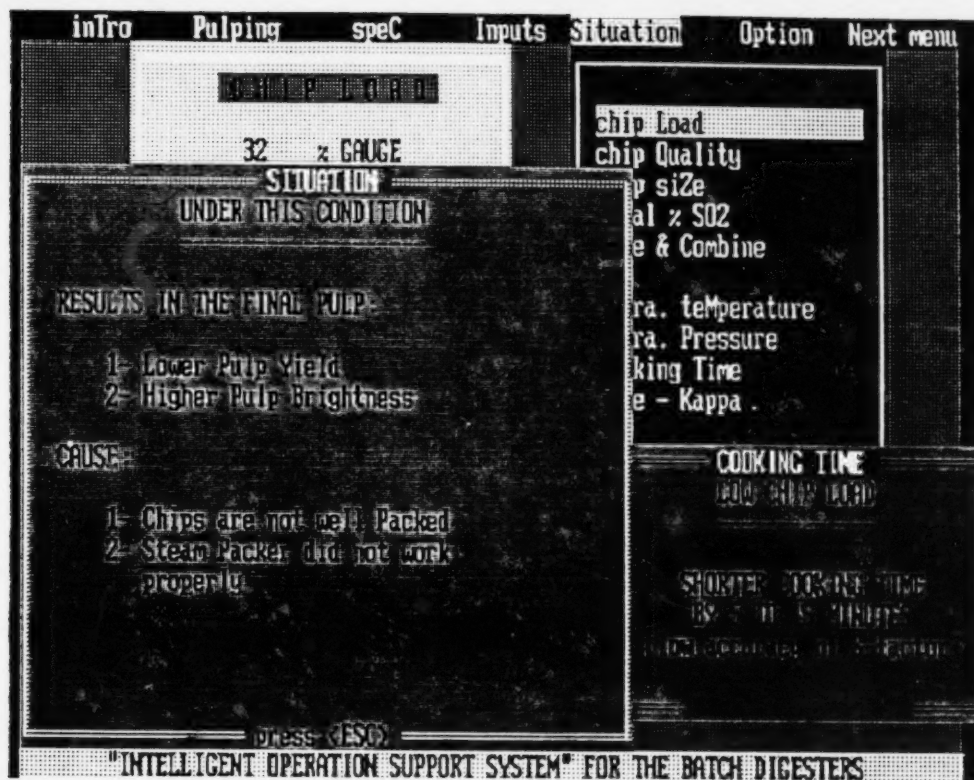


FIGURE 7.2 Display of qualitative reasoning results for "chip load"

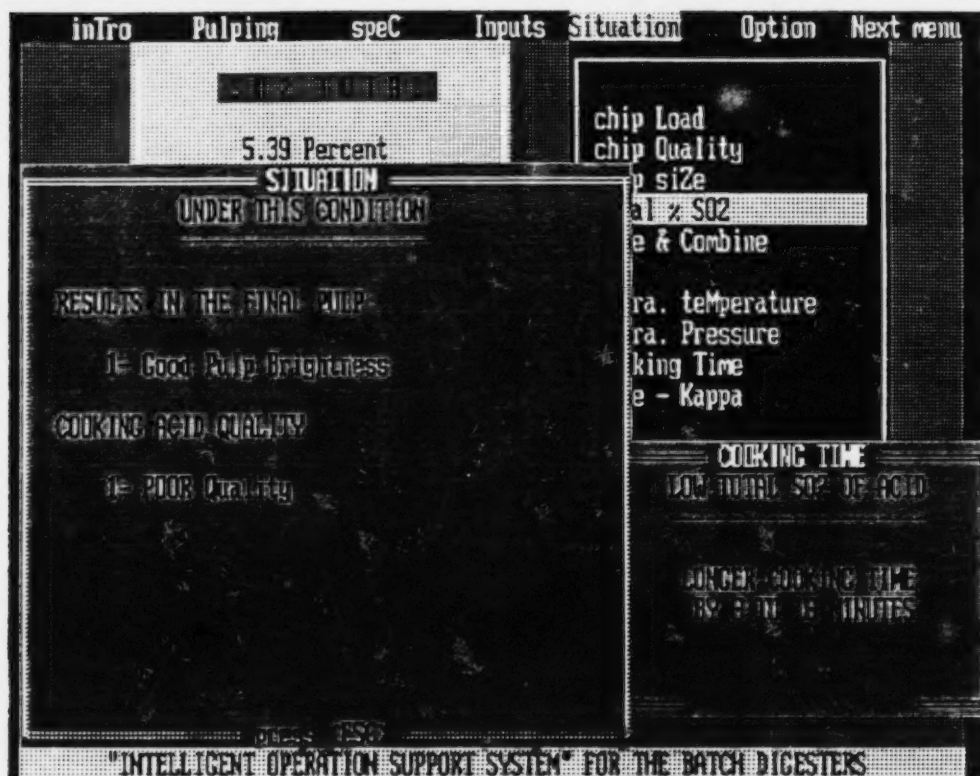


FIGURE 7.3 Display of qualitative reasoning results for "total SO₂"

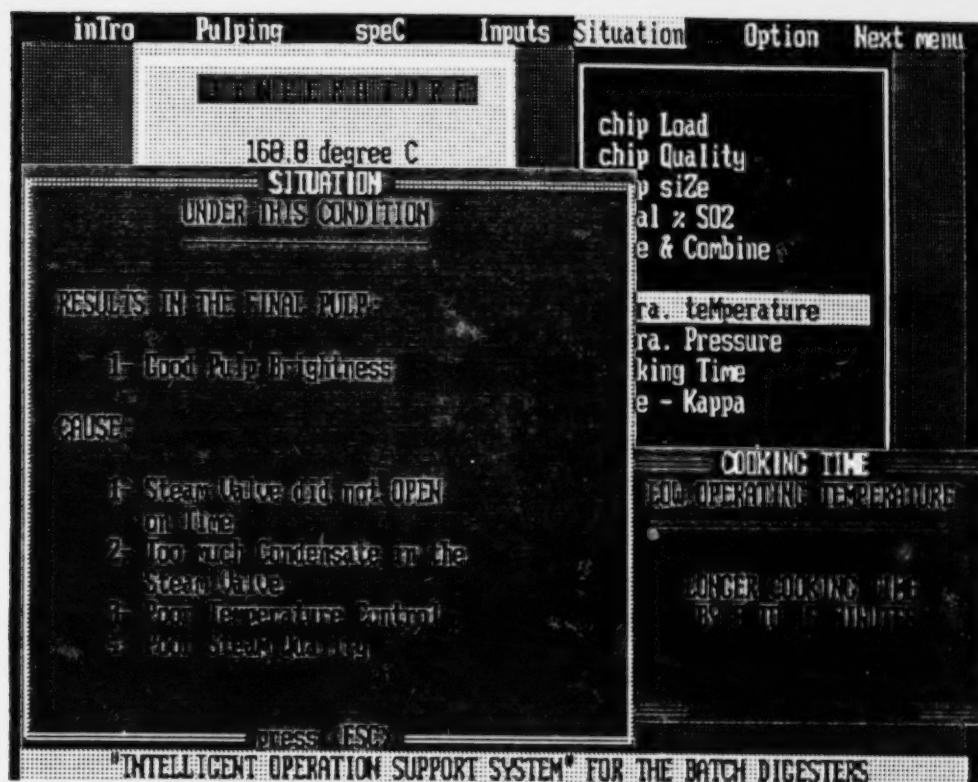


FIGURE 7.4 Display of qualitative reasoning results for "operating temperature"

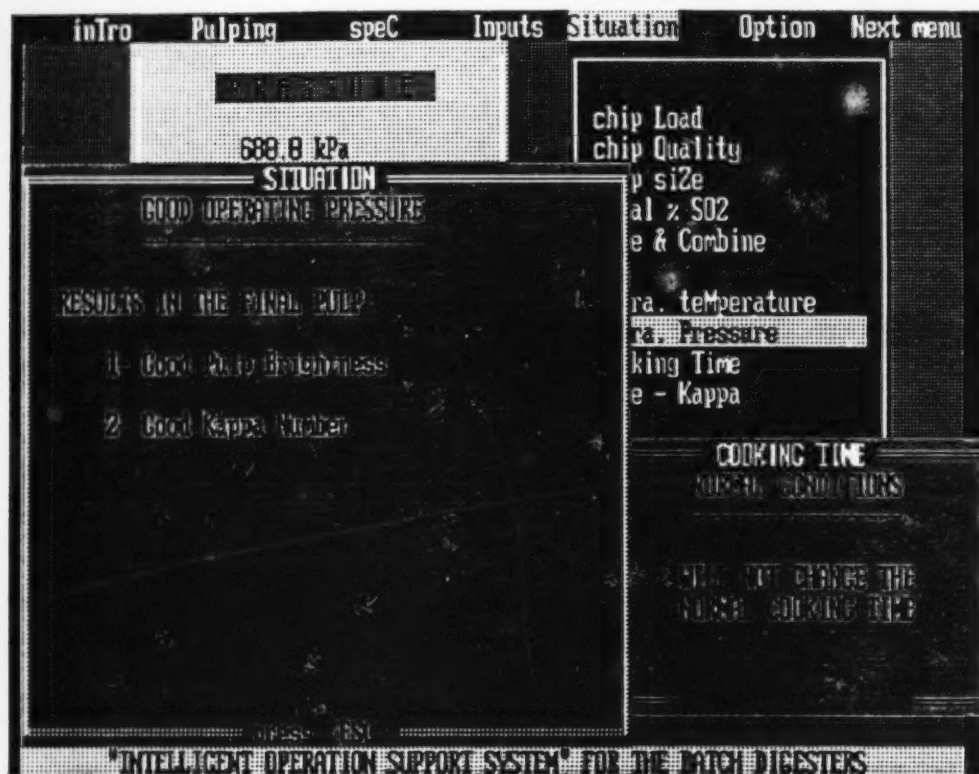


FIGURE 7.5 Display of qualitative reasoning results for "operating pressure"

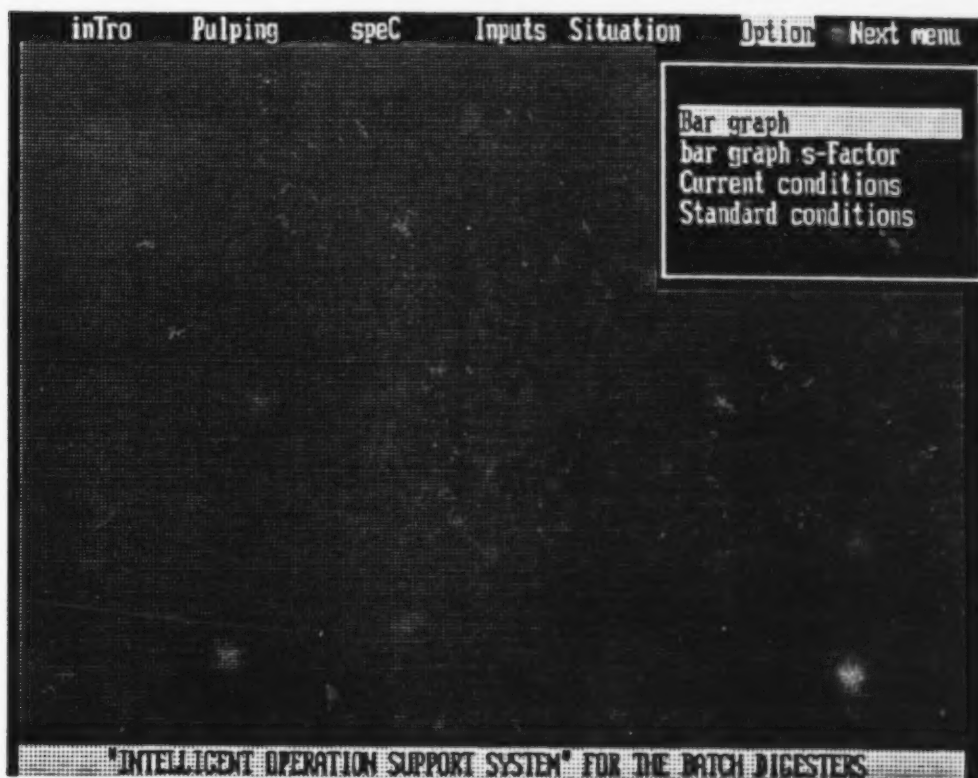


FIGURE 7.6 Selection for graphical display of "cooking times"

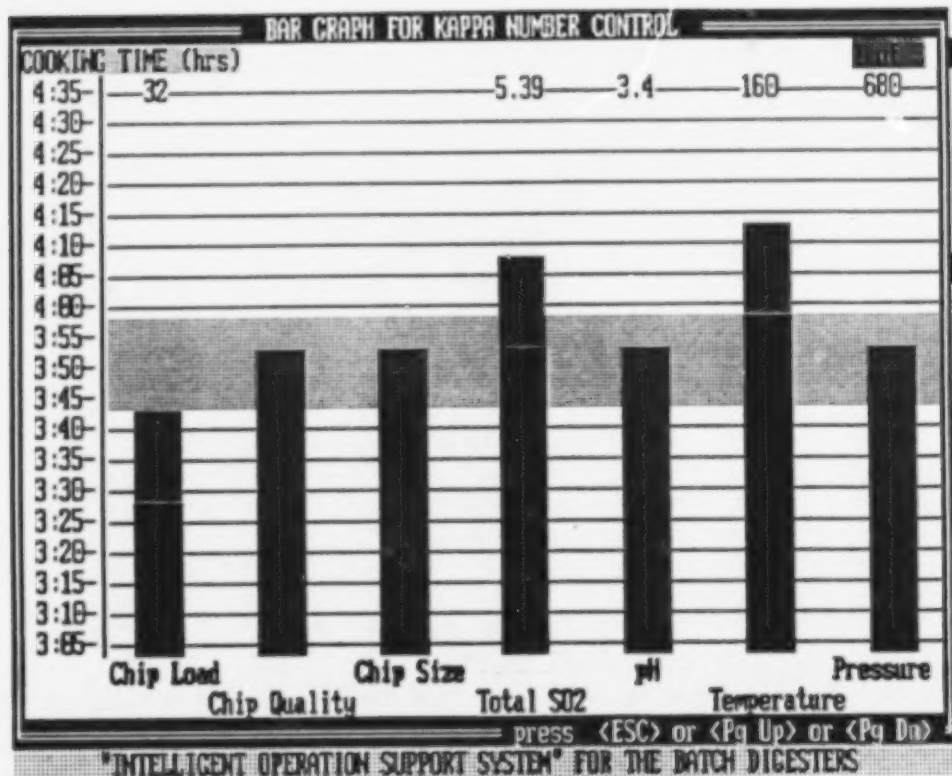


FIGURE 7.7 Estimated "cooking time" for each individual input variables (level I)

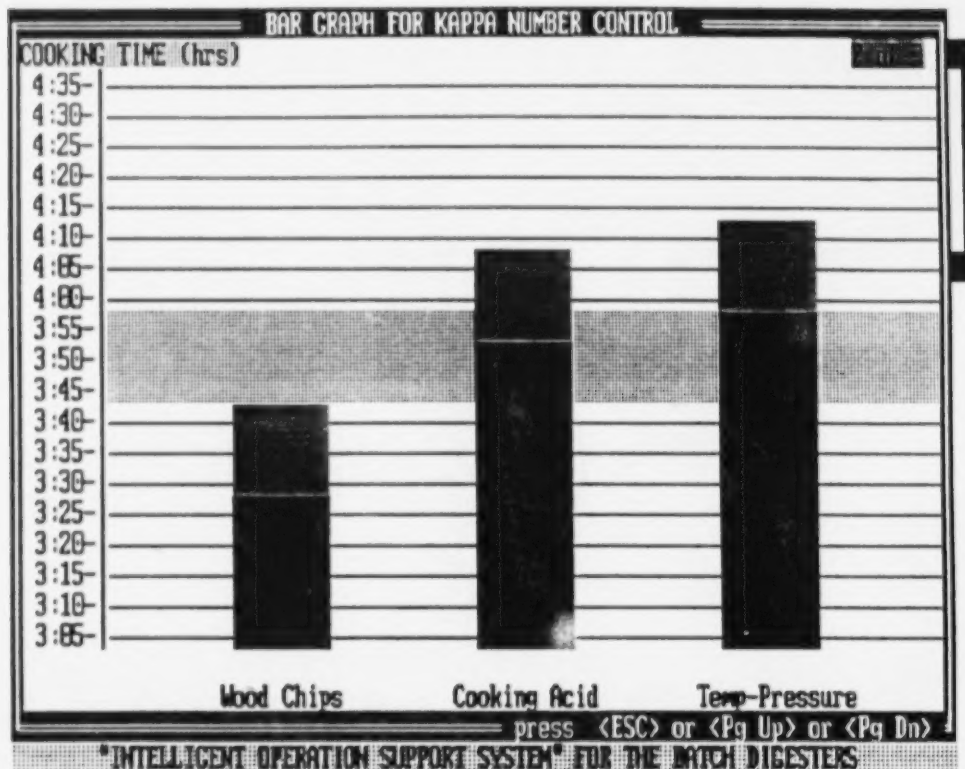


FIGURE 7.8 Estimated "cooking time" based on selected coupled input variables (Level II)

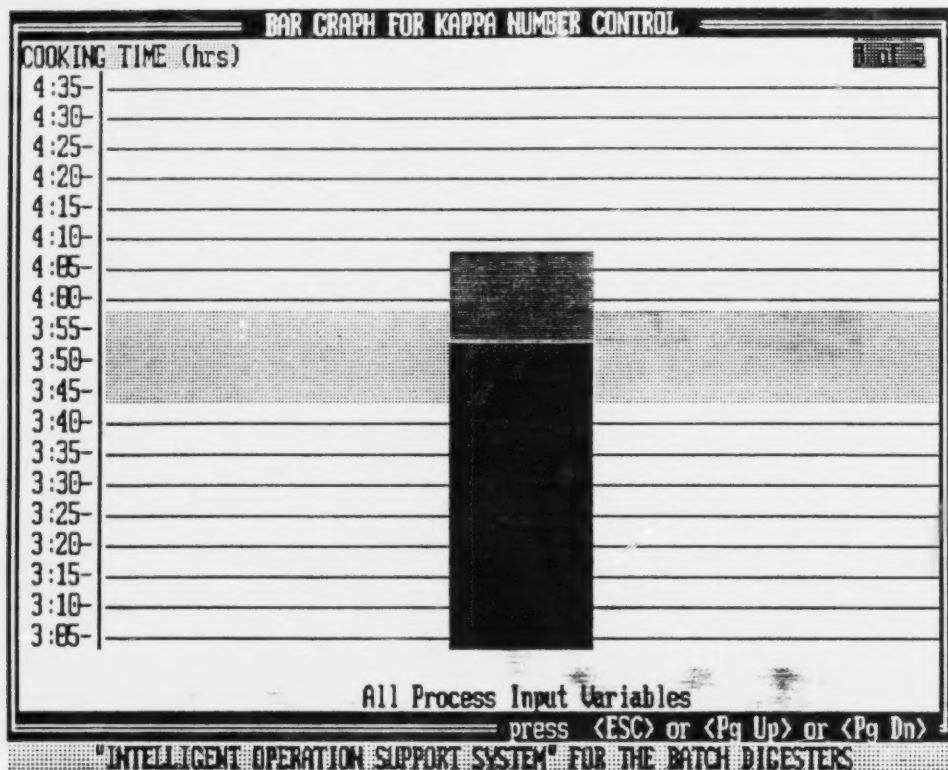


FIGURE 7.9 Estimated "cooking time" based on all input variables coupled together (Level III)

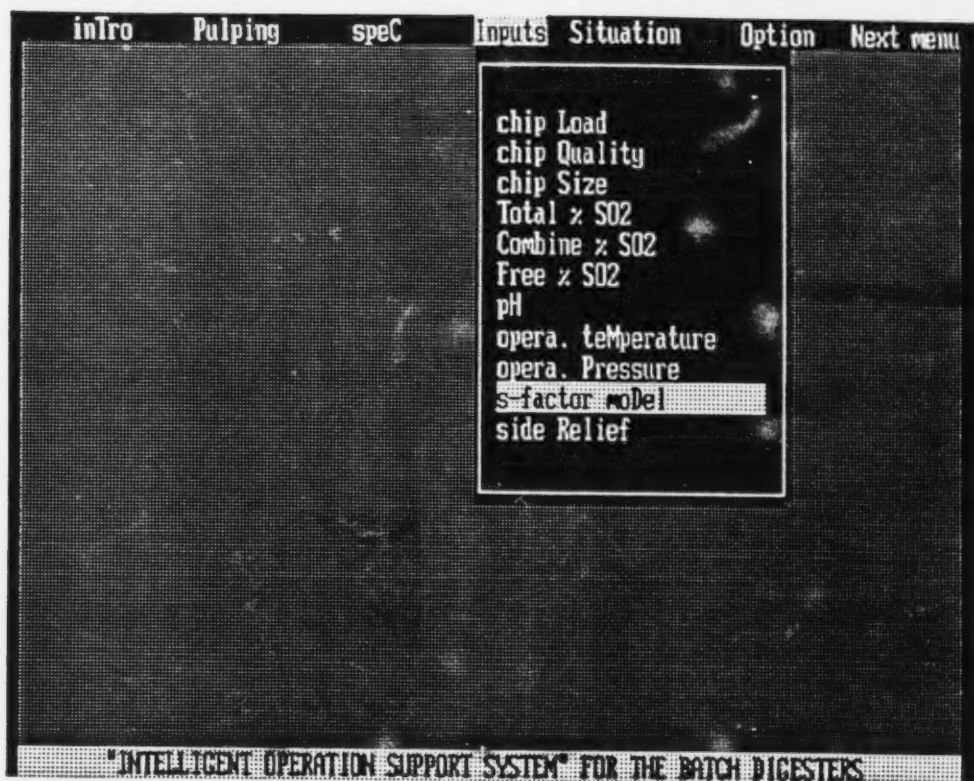


FIGURE 7.10 Selection to access to quantitative computation system

inTro	Pulping	speC	Inputs	Situation	Option	Next menu
COOKING MODEL						
Enter the % total SO ₂ of the cooking acid sample collected 10 minutes after the reaction temperature has reached a MAXIMUM[x.xx]						→ 1.89
Enter the TIME the sample was collected (i.e. time from the start of the cycle)[x.xx]						[hr.min] → 2:18
Enter the S-factor at that time .[xx.xx].or.[xxx.x]						→ 125
"INTELLIGENT OPERATION SUPPORT SYSTEM" FOR THE BATCH DIGESTERS						

FIGURE 7.11 Input data for quantitative computation system

inTro	Pulping	speC	Inputs	Situation	Option	Next menu
-------	---------	------	--------	-----------	--------	-----------

COOKING MODEL

Enter the α total SO₂ of the cooking acid sample
collected 10 minutes after the reaction temperature
has reached a MAXIMUM[x.xx] --> 1.89

Enter the TIME the sample was collected [hr.min]
(i.e. time from the start of the cycle)[x.xx] --> 2:18

Enter the S-factor at that time .[xx.xx].or.[xxx.x] --> 125.0

The S-FACTOR required is --> 761.59

press <ESC> to continue

"INTELLIGENT OPERATION SUPPORT SYSTEM" FOR THE BATCH DIGESTERS

FIGURE 7.12 Display of the required target "S_m-factor"

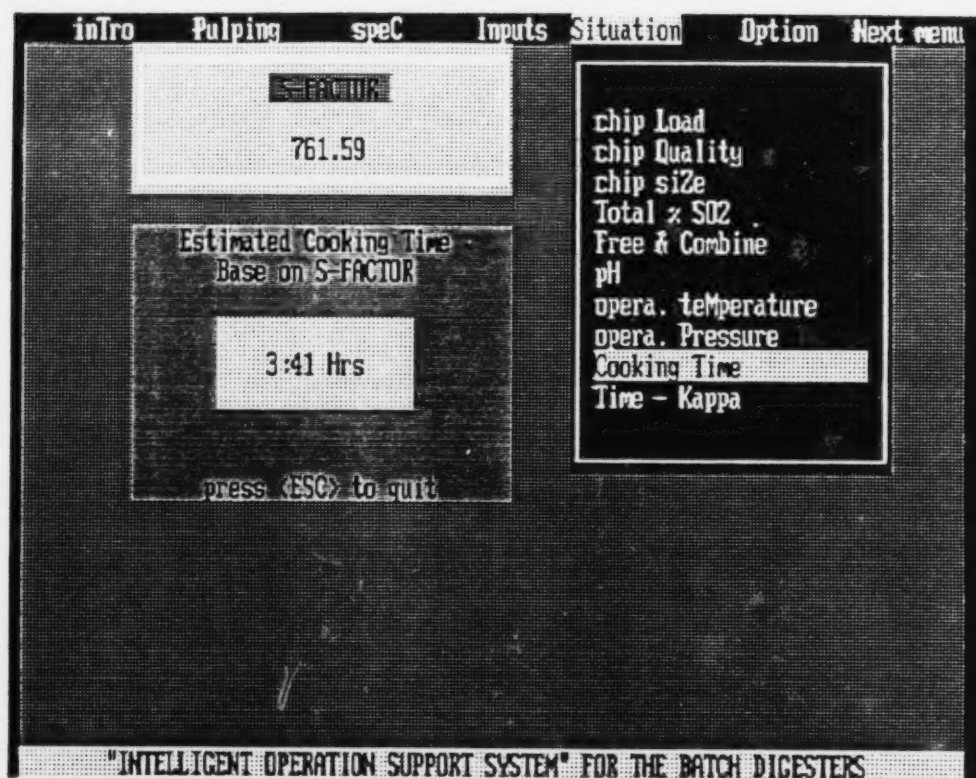


FIGURE 7.13 Display of the estimated "cooking time"

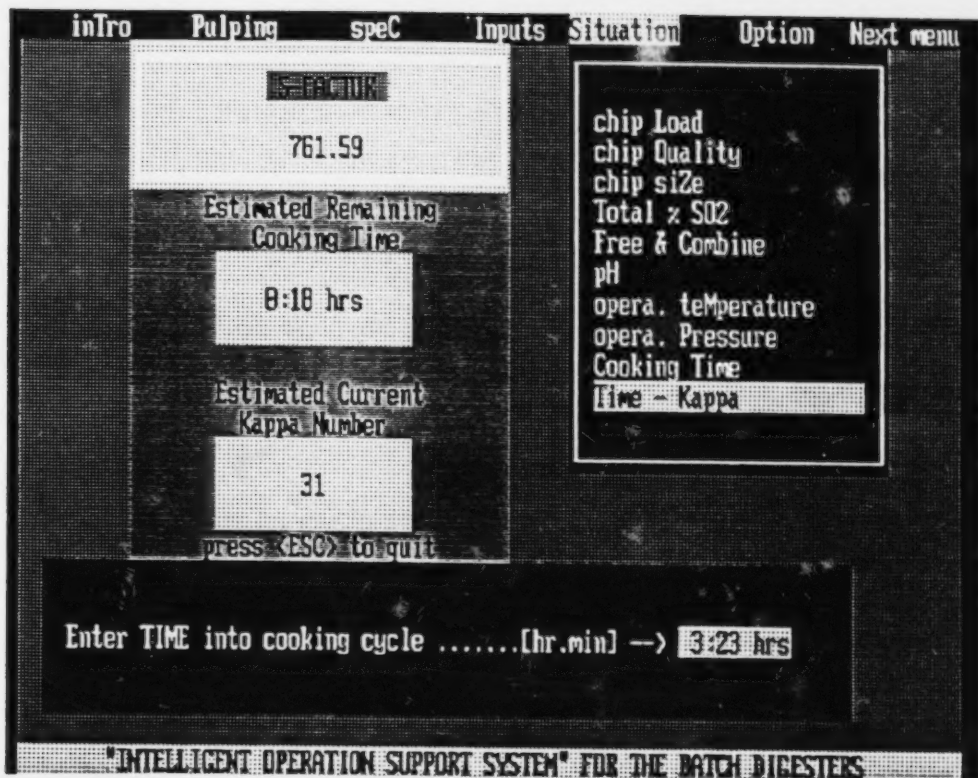


FIGURE 7.14 Display of the estimated "remaining cooking time" and "kappa number"

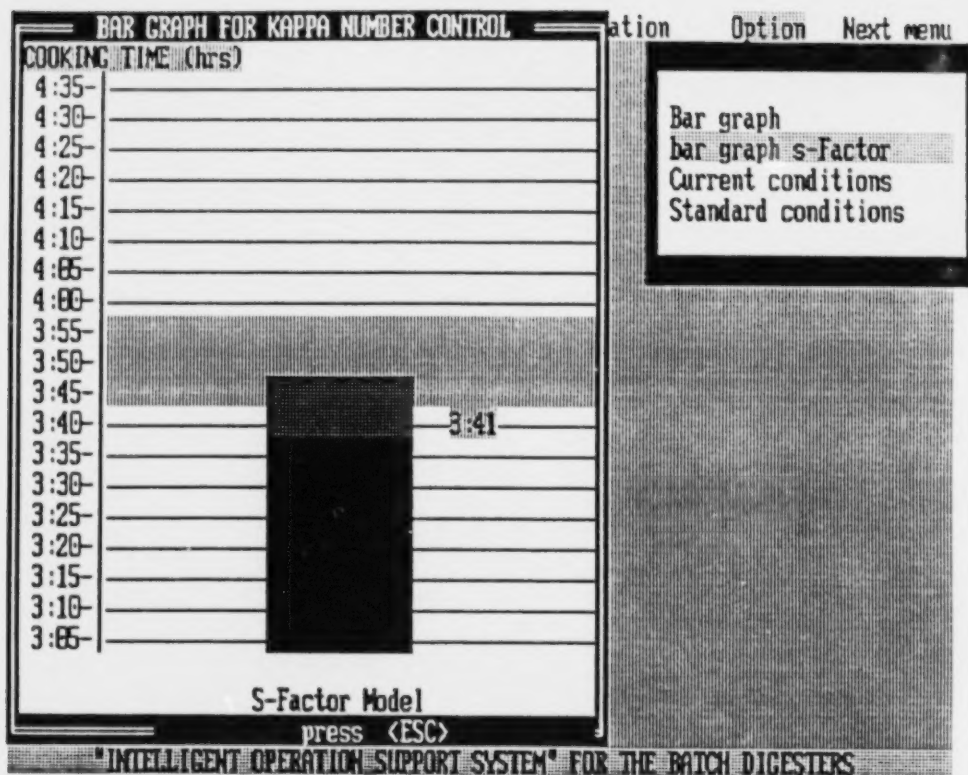


FIGURE 7.15 Graphical representation of the calculated total "cooking time"

CHAPTER 8

CONCLUSION

In this chapter, the author would like to conclude this thesis. The author feels that the following contributions were made to both scientific research and industrial applications during my thesis research project.

1. This "Intelligent Operation Support System" (IOSS) for batch sulphite pulping process is the first prototype system developed for the pulping industry. This original research will pave the way for a new research direction for future process operation and control in manufacturing industries.
2. This project is developed for a real industrial process. All the data and knowledge were collected from Fraser's sulphite pulping process. It was so designed that it could be directly employed in the plant. Evaluation at Fraser's mill is very satisfactory, and the project was greatly appreciated by industrial operators and engineers. Such a project aims at narrowing the gap between academic research and real industrial application.
3. The objective of this project is to provide assistance to human being to achieve better operation of the batch sulphite pulping process. Figure 8.1 demonstrates that this system could be used by different users. For example, the system can be used to provide operation support for operators, to help

engineers better understanding the pulping process, as well as to train new employees. It is a menu-driven system and running under a user-friendly environment. The IOSS intends to increase the interaction between the human operators and the pulping process by using both qualitative reasoning and quantitative computation.

4. Qualitative reasoning function helps operators to select the near-optimal operation conditions based on the initial process variables and operation conditions. It provides "cooking time" for the production of good quality pulp as well as faults diagnosis.

5 Based on the quantitative computation, the IOSS can provide on-line supervisory function under special process operation situations. It can alert the operator about the remaining "cooking time" before the "drain down" operation for a cooking cycle. It also estimate "kappa numbers" during the cooking cycle.

6. The development of an IOSS for a real industrial application requires the knowledge from different fields. In this project, the author was trained in an interdisciplinary research methodology. The author utilized the knowledge from the chemical engineering background, artificial intelligence (AI) technology, and process control.

7. During the construction of IOSS, many advanced AI techniques were investigated to improve system efficiency.

(i) The integration distribution intelligence system concept was implemented in IOSS. Both qualitative reasoning and quantitative computation were integrated together.

(ii) IOSS utilizes variable rules to organize its knowledge base. Such technique greatly reduces both memory space and running time.

(iii) IOSS has a common sense reasoning mechanism to prevent the system from processing impossible physical data. Such feature is implemented at the user interface level and protects the users from entering wrong input data.

(iv) A user friendly interface is built in IOSS. The interface provides many features such as graphic representation of both process flow chart and reasoning-computation results, window and menu-driven environment, natural language interface, ignoring wrong input data, multi-task environment. It is implemented using "C" language, running on PC/MSDOS that offers the user the lowest development cost, and the higher flexibility to modify the system.

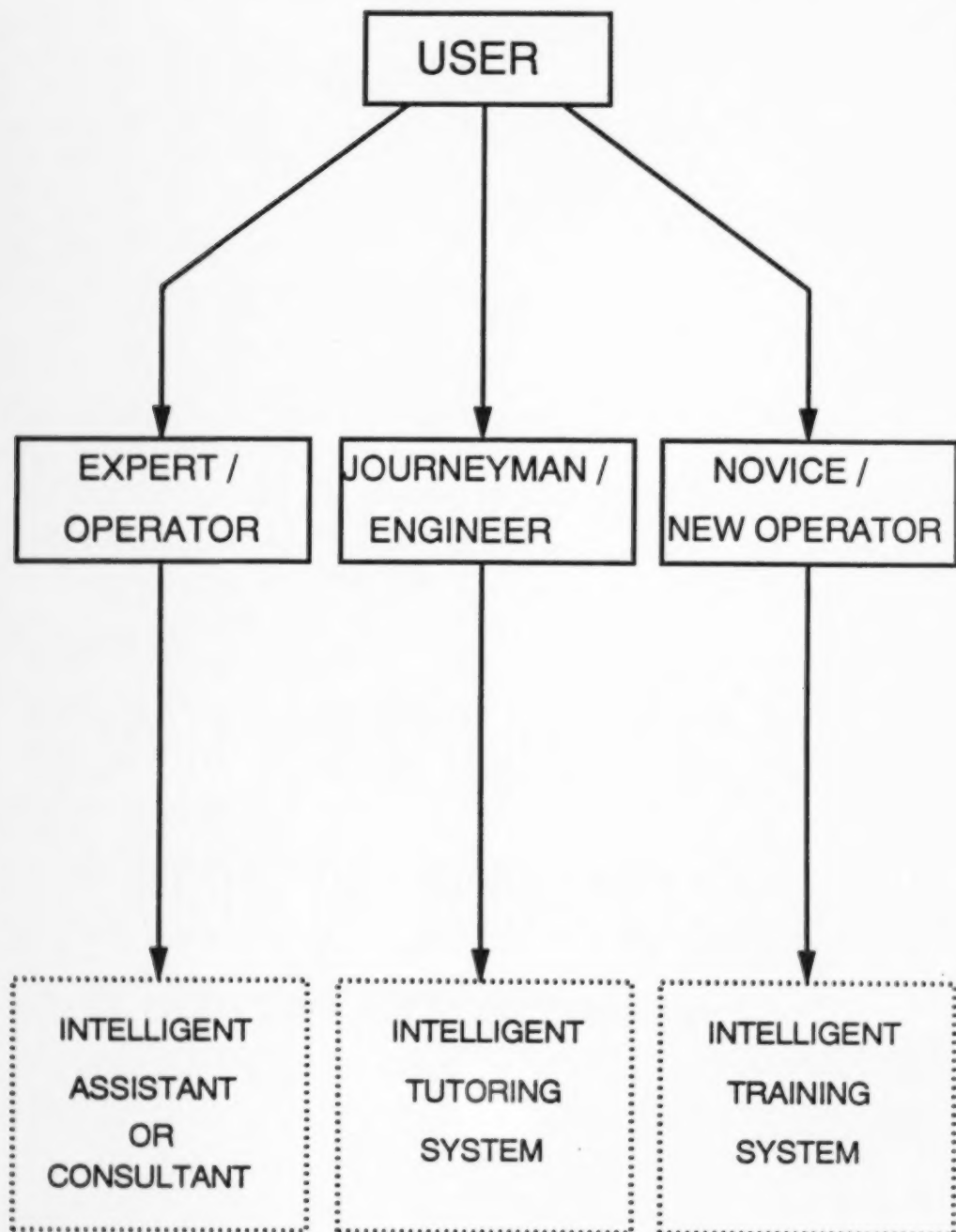


FIGURE 8.1 Types of users for IOSS

BIBLIOGRAPHY

- Andrews, M. Barnes, R. *Better-Quality Pulp Gained by Using Statistics to Analyze Chip Variation.* Pulp & Paper: 103-107, November 1985
- Azad, M.M. *The Role of Human Factors in Expert System Design and Acceptance.* Expert Systems: 395-414 30(4), 1988
- Balchen, J.G. Mumme, K. I. *Process Control - Structures and Applications.* Van Nostrand Reinhold Company, New York, 1988
- Beller, J. Kammerer, R.; Kaya, A.; Keyes, M.A. *Kappa Number Estimation, Control and Management in the Pulping Process.* American Control Conference: 1988
- Borsje, H.J. Christian, J.T.; Finn, G.A. *Integration of Expert systems with Plant Process Control Systems.* 40th Canadian Chemical Engineering Conference: July 1990
- Buck, L. *Human Operators and Real-Time Expert Systems.* Expert Systems: 227-237, November 1989
- Bylund, L. Thorsell, L. *On-Line Cooking Liquor Analyzer - A Means For Effective Control of Sulfite Digester.* Tappi Proceeding (International Sulphite Pulping Conference): 285-292, 1982
- Corbin, J. Rao, M. *An Intelligent Simulation System for Monitoring a Batch Pulping Process.* IACED: 221-223, Calgary, Alberta, April 8-10, 1991

- Egol, L. *Expert Systems Come to the Desktop*. Chemical Engineering: 157-159, March 1990
- Ferrada, J.J. Holmes, J.M. *Development of Expert Systems*. Chemical Engineering Progress: 34-41, April 1990
- Finn, G.A. Reinschmidt, K.F. *Expert Systems in Operation and Maintenance*. Chemical Engineering: 131-135, February, 1990
- Freedman, B.G. *Model Reference Control For Batch Digesters*. Chemical Engineering Progress: 82-86, April 1976
- Gidwani, K.K. *The Role Of Artificial Intelligence Systems in Process Control*. American Control Conference: 881-883, 1985
- Haywood, S.T. *An Empirical Cooking Model for Magnesium Bisulphite Pulp*. Pulp & Paper Canada: 112-114 90:6, 1989
- Hinrichs, D.D. *The Effect of Kraft Pulping Variables on Delignification*. Tappi: 173-175, Vol. 50, No. 4 April 1967
- Johnson, W.F. *EXPERT SYSTEMS: Their Application in the Canadian Transportation Sector*. Transport Canada: January 1986
- Kaemmerer, W.F. *Knowledge Representation Techniques for Object-Oriented Expert Systems*. ISA: 1990
- Kaemmerer, W.F. Christopherson, P.D. *Using Process Model with Expert Systems to Aid Process Control Operators*. American Control Conference: 892-897, 1985
- Kerr, A.J. *The Kinetics of Kraft Pulping - Batch Digester Control*. Tappi: 89-91, Vol. 59, No. 5, May 1976
- Kleppe, P.J. *Kraft Pulping*. Tappi Journal: 35-46, Vol. 53, No.1 January 1970

- Kramer, B. Liebowitz, H. *Intelligent Materials Processing Robotics & computer-Integrated Manufacturing: 141-145, Vol. 3, No. 2, 1987*
- Levesque, D.M. *Kappa Number Standard Deviation Reduction (project report to P.D. Carlisle) Fraser Inc. September, 1989*
- Lukas, M.P. Keyes, M.A. *The Evolution of Expert Systems for Process Control. Pulp & Paper Canada: T123-T127, 90:4, 1989*
- McCubbin, N. *The Basic Technology of the Pulp and Paper Industry and its Environmental Protection Practices. Environmental Protection Programs Directorate: October 1983*
- Parker, H.V. *Purchased Chip Quality Control Provides Improved Pulp Quality and Yield. Tappi Journal: December 1988*
- Patton, W.R. *BATCH vs CONTINUOUS: Which is Better for Pulping Kraft. Pulp & Paper: 70-72, May 1977*
- Rao, M. *A graduate course on Engineering Application of Artificial Intelligence.(submitted for publication) Chemical Engineering Education., 1991*
- Rao, M. Ying, Y.; Corbin, J. *Intelligent Engineering Approach to Pulp and Process Control. CPPA (77th Annual Meeting): 195-197, 1991*
- Rao, M. Jiang, T.S.; Tsai, J.J.-P. *Combining Symbolic and Numerical Processing for Real-Time Intelligent Control.Engineering Application of AI: 19-27, Vol. 2, March 1989*
- Rao, M. Tsai, J.J.-P.; Jiang, T.S. *An Intelligent Decisionmaker for Optimal Control. Applied Artificial Intelligence: 285-305, 1988*
- Shirley, R. *An Expert System for Process Control. Tappi Journal: 53-56, May 1987*

- Simmonds, W.H. *Future Trends in Process Control.* Electronics & Power: 570-572, September 1987
- Slowick, A.M. letter to Mr. Jean Corbin: *An Expert System for the operation of the Batch Digesters.* Fraser Paper, limited., September 18, 1991
- Smith, D. Hubert, B. *Digesters' Operation Manual* Fraser Inc. 1983
- Smook, G.A. *Handbook for Pulp and Paper Technologists.* TAPPI / CPPA: Chapter 1-4, 1982
- Stephanopoulos, G. *The Future of Expert Systems in Chemical Engineering.* Chem. Eng. Prog., Vol. 83, No. 44, 1987
- Surko, P. *Tips for Knowledge Acquisition.* PC AI: 14-18 May / June, 1989
- Vroom, K.E. *The "H" Factor: A Means Of Expressing Cooking Times and Temperature as a Single Variable.* Pulp and Paper Magazine of Canada (Convention Issue): 228-231, 1957
- Volpe, G. Letter to Dr. Ming Rao *Digesters Control - Expert System*". Fraser Inc, September, 12th, 1991
- Wang, X. *A new Method for Determining the Activation Energy. of the Delignification Reaction.* Tappi Journal: 125-127, September 1987
- Waye, D. Terroux, A. *SIMSMART: Synthetic Intelligence(SI): The integration of AI and Simulation.* Pulp & Paper Canada: 83-87, 92:1, 1991
- Waye, D. *SIMSMART: Dynamic Simulation for Engineering Design, Operator training, of Industrial Process and Automated Control.* Tappi Journal: 52-56, Aug, 1987
- West, M.Z. Peacock, T.E. *Kappa Number Revisited for Application in Computer Programs.* Tappi Journal: P. 196, July 1987

- Whitehead, J.D. Roach, J.W. *HOIST: A Second-Generation Expert System Based on Qualitative Physics.* AI Magazine: 108-119, Fall 1990
- Whitley, J.C. *Statistic Improve K Number Control at Federal Paper Board Pulp Mill.* Pulp & Paper: 99-102, January 1987
- Yorston, F.H. Liebergott, N. *Correlation of the Rate of Sulphite Pulping with Temperature and Pressure. Further Studies..* Pulp & Paper Magazine of Canada: T272-T278, May 1965

APPENDIX

SYSTEM SOURCE CODE

	PAGE
1. DIGESTER.EXE (C language)	
INPUTS.C	140
KNBASE.C	150
MAINPRO.C	182
MMODEL.C	187
OPTIONS.C	191
READDATA.C	200
READTXT.C	201
RULEBASE.C	202
SIDEFUNC.C	203
SITUATIO.C	205
SPEC.C	213
STARSHUT.C	222
2. COVER.EXE (C language)	
PAGE1.C	226
3. FLOW.EXE (C language)	
FLOWCHAR.C	228
SITUCHIP.C	237
4. COOKMODL.EXE (FORTRAN language)	
COOKMODL.FOR	238

```

// *****
//
//          MODULE: "INPUTS.C"
//
// *****
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

float windfunc();
float wind2func();
char numfunc(int s);
char *size_selection(float);
char *quality_selection(float);

extern float database[10];
extern char shadow;
extern float s_fact[4];

intelligent(char index1)
{
    switch(index1){

        case 'L':
            load_func();
            break;

        case 'Q':
            quality_func();
            break;

        case 'S':
            size_func();
            break;

        case 'T':
            total_func();
            break;

        case 'C':
            combine_func();
            break;

        case 'F':
            free_func();
            break;

        case 'H':
            ph_func();
            break;

        case 'M':
            temp_func();
            break;
    }
}

```



```

        case 'P':
            press_func();
            break;

        case 'D':
            cooking_model();
            break;

        case 'R':
            relief_func();
            break;

    }
    return;
}

/* ----- C H I P L O A D ----- */

load_func()
{
    char *select, *unit, *titl, *ques, *loadcond;
    float loadl;
    int maxdig, minl, maxl;

    {
        select=
            "ENTER CHIP LOAD VALUE -->";

        unit=
            " % GAUGE";

        titl=
            " C H I P L O A D ";

        ques=
            "What is the chip load in Percentage in the\n"
            " digester at the beginning of the cook ? ";

        maxdig = 1;

        minl = 20;
        maxl = 55;
    }

    {
        loadl = windfunc(titl,ques,select,unit,maxdig,minl,maxl);
        if(loadl == 999) return;
        database[0] = loadl;
        rwind();
    }
    return;
}

/* ----- C H I P Q U A L I T Y ----- */

quality_func()
{
    int varl;

    mwind(2,2,51,17,attr(0,1),attr(YELLOW,BLACK),1);
    wtitle(CENTER,TOP,attr(RED,BLACK)," C H I P Q U A L I T Y ");
    pwindfc(4,attr(14,1),"[1] GOOD QUALITY");
}

```

```

pwindfc(5,attr(14,1),"[2] OLD CHIPS");
pwindfc(6,attr(14,1),"[3] MORE 10% HARDWOOD");
pwindfc(7,attr(14,1),"[4] OLD CHIPS & MORE 10% HARDWOOD");
mbox(8,6,46,11,9,2);
pwindfc(1,attr(15,1),"What is the average chip QUALITY in the");
pwindfc(2,attr(15,1),"digester at the beginning of the cook ?");
pwindf(3,11,attr(15,1),"ENTER SELECTION 1, 2, 3 OR 4 -->");
pwindf(38,11,attr(0,7),"");
mbox(3,12,50,16,5,2);
setcursor(ON);
gowindcr(39,11);
for(;;){
    var1 = keyin();
    if(var1==49 || var1==50 || var1==51 || var1==52) break;
    if(var1==ESC) {
        rwind();
        return;
    }
    setcursor(OFF);
    var1 = var1-48.0;
    database[1] = var1;
    pwindf(39,11,attr(0,7),"%hu",var1);
    waitkey(PRET);
    rwind();
}
return;
}

/* ----- C H I P S I Z E ----- */

size_func()
{
    int var1;

    mwind(2,2,51,16,attr(0,1),attr(YELLOW,BLACK),1);
    wtitle(CENTER,TOP,attr(RED,BLACK)," C H I P S I Z E ");
    pwindfc(4,attr(14,1),"[1] STANDARD SIZE");
    pwindfc(5,attr(14,1),"[2] UNDER STANDARD SIZE");
    pwindfc(6,attr(14,1),"[3] OVER STANDARD SIZE");
    mbox(12,6,42,10,9,2);
    pwindfc(1,attr(15,1),"What is the average chip SIZE in the");
    pwindfc(2,attr(15,1),"digester at the beginning of the cook ?");
    pwindf(5,10,attr(15,1),"ENTER SELECTION 1, 2, OR 3 -->");
    pwindf(38,10,attr(0,7),"");
    mbox(3,11,50,15,5,2);
    setcursor(ON);
    gowindcr(39,10);
    for(;;){
        var1 = keyin();
        if(var1 == 49 || var1 == 50 || var1 == 51) break;
        if(var1 == ESC) {
            rwind();
            return;
        }
    }
    setcursor(OFF);
    var1 = var1-48.0;
    database[2] = var1;
    pwindf(39,10,attr(0,7),"%hu",var1);
    waitkey(PRET);
    rwind();
}
return;
}

```

```

/* ----- S O 2   T O T A L   ----- */
total_func()
{
    char *select, *unit, *titl, *ques, *totalcond;
    float totall;
    int maxdig;
    double minl,maxl;

    select=
        " ENTER TOTAL SO2  -->";

    unit=
        "Percent";

    titl=
        " S O 2   T O T A L ";

    ques=
        "What is the total SO2 of cooking acid\n"
        " in percent during acid filling ?  ";

    maxdig = 3;
    minl = 3.5;
    maxl = 7.5;

    totall = wind2func(titl,ques,select,unit,maxdig,minl,maxl);
    if(totall == 999) return;
    database[3] = totall;
    rwind();
    s_fact[0]=0.0;
return;
}

/* ----- S O 2   C O M B I N E   -----
*/
combine_func()
{
    char *select, *unit, *titl, *ques, *combcond;
    float comb1;
    int maxdig;
    double minl,maxl;

    {
        select=
            " ENTER COMBINE SO2  -->";

        unit=
            "Percent";

        titl=
            " S O 2   C O M B I N E ";

        ques=
            "What is the combine SO2 of cooking acid\n"
            " in percent during acid filling ?  ";

        maxdig = 3;
        minl = 2.0;
        maxl = 4.0;
    }
}

```

```

        comb1 = wind2func(tit1,ques,select,unit,maxdig,min1,max1);
        if(comb1 == 999) return;
        database[7] = comb1;
        rwind();
    }
    return;
}

/* ----- S O 2 F R E E ----- */

free_func()
{
    char *select, *unit, *tit1, *ques, *freecond;
    float free1;
    int maxdig;
    double min1,max1;
    {
        select=
            " ENTER FREE SO2 -->";
        unit=
            " Percent";
        tit1=
            " S O 2 F R E E ";
        ques=
            "What is the free SO2 of cooking acid\n"
            " in percent during acid filling ? ";

        maxdig = 3;
        min1 = 2.0;
        max1 = 4.0;
    }
    {
        free1 = wind2func(tit1,ques,select,unit,maxdig,min1,max1);
        if(free1 == 999) return;
        database[8] = free1;
        rwind();
    }
    return;
}

/* ----- P H ----- */

ph_func()
{
    char *select, *unit, *tit1, *ques;
    char *phcond, *filem;
    char *readstuff();
    char *get_knowledge();
    float ph1;
    int maxdig;
    double min1,max1;
    {
        select=
            " ENTER THE pH VALUE -->";
        unit=
            "";
        tit1=
            " p H of A C I D ";
        ques=
            "What is the pH value of cooking\n"
            " acid during acid filling ? ";

        maxdig = 2;
        min1 = 2.80;
        max1 = 5.00;
    }

```

```

    }
    {
        ph1 = wind2func(tit1,ques,select,unit,maxdig,min1,max1);
        if(ph1 == 999) return;
        database[4] = ph1;
        rwind();
    }
return;
}

/* ----- T E M P E R A T U R E ----- */

temp_func()
{
    char *select, *unit, *tit1, *ques, *tempcnd;
    float temp1;
    int maxdig,min1,max1;
    {
        select=
            " ENTER TEMPERATURE -->";
        unit=
            " degree C";
        tit1=
            " T E M P E R A T U R E ";
        ques=
            "What is the operating temperature\n"
            " in degree C in the digester \n"
            " during the cooking cycle ? ";
        maxdig = 2;
        min1 = 155;
        max1 = 175;
    }
    {
        temp1 = windfunc(tit1,ques,select,unit,maxdig,min1,max1);
        if(temp1 == 999) return;
        database[5] = temp1;
        rwind();
    }
return;
}

/* ----- P R E S S U R E ----- */

press_func()
{
    char *select, *unit, *tit1, *ques, *prescnd;
    float press1;
    int maxdig,min1,max1;
    {
        select=
            " ENTER PRESSURE -->";
        unit=
            " kPa";
        tit1=
            " P R E S S U R E ";
        ques=
            "What is the OPERATING PRESSURE\n"
            " in kPa the digester \n"
            " during the cooking cycle ? ";
        maxdig = 2;
        min1 = 500;
        max1 = 800;
        press1 = windfunc(tit1,ques,select,unit,maxdig,min1,max1);
    }
}

```

```

        if(press1 == 999) return;
        database[6] = press1;
        rwind();
    }
return;
}

/* ----- W I N D O W S ----- */

float windfunc(char *a, char *b, char *c, char *d, int e, int f, int g)
{
    char chl[5];
    int i, val, x1, x2;

    shadow=TRUE;
    mwind(1,2,47,16,attr(0,1),attr(YELLOW,7),1);
    wtitle(CENTER,TOP,attr(RED,7),a);
    shadow=FALSE;
    pwindfc(1,attr(15,1),"%s",b);
    pwindf(1,10,attr(15,1),"%s",c);
    pwindf(30,10,attr(15,1),"%s",d);
    mbox(2,11,46,15,5,3);
    val = 999;
    while(val < f || val > g)
    {
        chl[0]='.';
        chl[1]='.';
        chl[2]='.';
        chl[3]='.';
        chl[4]='.';
        setcursor(ON);
        gowindcr(26,10);
        pwindf(26,10,attr(0,7),"  ");
        for(i=0; i<=e; i++)
        {
            for(;;)
            {
                x1 = keyin();
                if(x1 == ESC) {
                    setcursor(OFF);
                    rwind();
                    return 999;
                }
                if(x1 == BS && i > 0)
                {
                    pwindf(25+i,10,attr(0,7)," ");
                    gowindcr(25+i,10);
                    i--;
                }

                if(x1 >= 48 && x1 <= 58) break;
                if(x1!=8) sound_func(4000);
            }
            x1 = x1-48;
            chl[i] = numfunc(x1);
            pwindf(26+i,10,attr(0,7),"%c",chl[i]);
            gowindcr(27+i,10);
        }
        val = atoi(chl);
        setcursor(OFF);
        pwindf(26,10,attr(0,15),"  ");
        pwindf(26,10,attr(0,15),"%d",val);
    }
}

```

```

if(val < f || val > g){
    sound_func(900);
    pwindfc(5,attr(14,1),"VALUE MUST BE BETWEEN ");
    pwindfc(6,attr(14,1),"          &          ");
    pwindf(16,6,attr(15,1),"d",f);
    pwindf(26,6,attr(15,1),"d",g);
    pwindfc(7,attr(4,15),"press <esc> to continue ");

    for(;;)
        { if ((x1=keyin()) == ESC)
            { pwindfc(5,attr(15,1),"
                pwindfc(6,attr(15,1),"
                pwindfc(7,attr(15,1),"
                break;
            }
        }
    }
    wtitle(CENTER,BOTTOM,attr(BLACK,7),"press ENTER to continue");
    enter_key();
return val;
}

/* ----- W I N D O W 2 ----- */

float wind2func(char *a, char *b, char *c, char *d, int e, double f,
double g)
{
    char chl[3];
    int i, x1, x2, px;
    double val;

    shadow=TRUE;
    mwind(1,2,47,16,attr(0,1),attr(YELLOW,7),1);
    wtitle(CENTER,TOP,attr(RED,7),a);
    shadow=FALSE;
    pwindfc(1,attr(15,1),"s",b);
    pwindf(1,10,attr(15,1),"s",c);
    pwindf(34,10,attr(15,1),"s",d);
    mbox(2,11,46,15,5,3);
    val = 999;
    px = 26;
    while(val < f || val > g)
    {
        setcursor(ON);
        gowindcr(px,10);
        pwindf(px,10,attr(0,7),"
        for(i=0; i<=e; i++)
        {
            for(;;)
            {
                x1 = keyin();
                if(x1 == ESC) {
                    setcursor(OFF);
                    rwind();
                    return 999;
                }
                if(x1 == BS && i > 0)
            {

```



```

        pwindf((px-1)+i,10,attr(0,7)," ");
        gowindcr(25+i,10);
        i--;
    }

    if(x1 >= 48 && x1 <= 58) break;
    if(x1==46) break;
    if(x1!=8) sound_func(4000);
}
x1 = x1-48;
chl[i] = numfunc(x1);
pwindf(26+i,10,attr(0,7),"c",chl[i]);
gowindcr(27+i,10);
}
val = atof(chl);
setcursor(OFF);
pwindf(26,10,attr(0,12),"l.2f",val);
if(val < f || val > g){

    sound_func(900);
    pwindfc(5,attr(14,1),"VALUE MUST BE BETWEEN ");
    pwindfc(6,attr(14,1),"          &          ");
    pwindf(16,6,attr(15,1),"l.2f",f);
    pwindf(26,6,attr(15,1),"l.2f",g);
    pwindfc(7,attr(4,15),"press <esc> to continue ");

    for(;;)
        { if ((x1=keyin()) == ESC)
            { pwindfc(5,attr(15,1),"          ");
              pwindfc(6,attr(15,1),"          ");
              pwindfc(7,attr(15,1),"          ");
              break;
            }
        }
    }
    wtitle(CENTER,BOTTOM,attr(BLACK,7),"press ENTER to continue");
    enter_key();
return val;
}

char numfunc(int s)
{ char t;

    if(s==1) t = '1';
    if(s==2) t = '2';
    if(s==3) t = '3';
    if(s==4) t = '4';
    if(s==5) t = '5';
    if(s==6) t = '6';
    if(s==7) t = '7';
    if(s==8) t = '8';
    if(s==9) t = '9';
    if(s==0) t = '0';
    if(s== -2) t = '.';
    return t;
}

char *quality_selection(float xx)
{
    char *select;

```

```

        if (xx == 1.0)
            select="Good Quality";
        if (xx == 2.0)
            select="Old Chips";
        if (xx == 3.0)
            select="More 10% Hardwood";
        if (xx == 4.0)
            select="Old & More 10% Hardwood";
    return select;
}

char *size_selection(float xx)
{
    char *select;
    if (xx == 1.0)
        select="Standard Size";
    if (xx == 2.0)
        select="Small Chips";
    if (xx == 3.0)
        select="Large Chips";
    return select;
}

```

```

// *****
//
//          MODULE: "KNBASE.C"
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

rule_base_setA(float ,float [10][2]);
rule_base_setB(float ,float [10]);

int ph_range(float);
char *ph_status(int);
char *ph_time_required(int);
char *ph_situation(int);
char *ph_time_limits(int, int);
float ph_weight(int);

int temp_range(float);
char *temp_status(int);
char *temp_time_required(int);
char *temp_situation(int);
char *temp_time_limits(int, int);
float temp_weight(int);

int press_range(float);
char *press_status(int);
char *press_time_required(int);
char *press_situation(int);
char *press_time_limits(int, int);
float press_weight(int);

int total_range(float);
char *total_status(int);
char *total_time_required(int);
char *total_situation(int);
char *total_time_limits(int, int);
float total_weight(int);

int load_range(float);
char *load_status(int);
char *load_time_required(int);
char *load_situation(int);
char *load_time_limits(int, int);
float load_weight(int);

int size_range(float);
char *size_status(int);
char *size_time_required(int);
char *size_situation(int);
char *size_time_limits(int, int);
float size_weight(int);

int quality_range(float);

```

```

char *quality_status(int);
char *quality_time_required(int);
char *quality_situation(int);
char *quality_time_limits(int, int);
float quality_weight(int);

int freecom_range(float, float);
char *freecom_status(int);
char *freecom_time_required(int);
char *freecom_situation(int);

char *couple_time_limit[4][2], *ul, *ll;
float couple_weight[4];
void weight_table(float);
void all_couple(void);
void load_quali_size_couple(float, float, float);
void ph_total_couple(float, float);
void press_temp_couple(float, float);
/* ***** */
/* ***** pH ***** */
/* ***** */

int ph_range(float ph)
{
    int loc;
    // Definition of Ranges of "ph" of cooking acid for rulebase

    // pH RANGES SETUP
    // -----

    float ranges[][2]=
        {(2.5,2.9), // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
         (3.0,3.3), // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
         (3.4,3.6), // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
         (3.7,4.0), // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
         (4.1,4.6), // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
         (0.0,0.0), // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
         (0.0,0.0)}; // Situation #6 (LOW_limit[][0],HIGH_limit[][1])
    // {LOW,HIGH}
    loc = rule_base_setA(ph, ranges);
    return loc;
}

char *ph_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* pH STATUS SETUP */
    /* -----*/

    status[0]="LOW pH OF ACID\n"
              "-----";
    status[1]="LOW pH OF ACID\n"
              "-----";
    status[2]="NORMAL CONDITIONS\n"
              "-----";
    status[3]="HIGH pH OF ACID\n"
              "-----";
    status[4]="HIGH pH OF ACID\n"
              "-----";

    status[5]=" ";

```

```

status[6]=" ";
return status[loc];
}

char *ph_time_required(int loc)
{
    char *time_required[10];

    /* pH TIME_REQUIRED SETUP */
    /* -----*/

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="SHORTER COOKING TIME\n"
        "BY 10 TO 20 MINUTES\n"
        "[low accuracy of S-factor]";

    time_required[1]="SHORTER COOKING TIME\n"
        "BY 0 TO 10 MINUTES\n"
        "[low accuracy of S-factor]";

    time_required[2]="WILL NOT CHANGE THE\n"
        "NORMAL COOKING TIME";

    time_required[3]="LONGER COOKING TIME\n"
        "BY 0 TO 10 MINUTES\n"
        "[low accuracy of S-factor]";

    time_required[4]="LONGER COOKING TIME\n"
        "BY 10 TO 20 MINUTES\n"
        "[low accuracy of S-factor]";

    time_required[5]=" ";
    time_required[6]=" ";

    return time_required[loc];
}

char *ph_situation(int loc)
{
    char *situation[10];

    /* pH SITUATION SETUP */
    /* -----*/

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Low Pulp Brightness \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "      1- High Acidity of Acid \n"
        " ";

```

```

situation[1]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Low Pulp Brightness       \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- High Acidity of Acid     \n"
" ";

situation[2]=
"          GOOD pH OF COOKING ACID      \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
"          2- Good Kappa Number         \n"
"          \n"
" ";

situation[3]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- LOW Acidity of Acid       \n"
"          2- Poor Cooking Acid Quality \n"
"          \n"
" SUGGESTION:                          \n"
"          \n"
"          1- Increase Operating Pressure Set Point";

situation[4]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- LOW Acidity of Acid       \n"
"          2- Poor Cooking Acid Quality \n"
"          \n"
" SUGGESTION:                          \n"
"          \n"
"          1- Increase Operating Pressure Set Point";

situation[5]=" ";
situation[6]=" ";

```

```

return situation[loc];
}

char *ph_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* pH TIME_LIMITS SETUP */
    /* -----*/

    // the pH high_time[0] and low_time[1] limits

    time_limits[0][0]="-10";
    time_limits[0][1]="-20";

    time_limits[1][0]="-0";
    time_limits[1][1]="-10";

    time_limits[2][0]="GOOD";
    time_limits[2][1]="GOOD";

    time_limits[3][0]="+10";
    time_limits[3][1]="+0";

    time_limits[4][0]="+20";
    time_limits[4][1]="+10";

    time_limits[5][0]="GOOD";
    time_limits[5][1]="GOOD";

    time_limits[6][0]="GOOD";
    time_limits[6][1]="GOOD";

    return time_limits[loc][hl];
}

float ph_weight(int loc)
{
    float weight[10];

    /* pH WEIGHT SETUP */
    /* -----*/
    // the pH weight under this conditions

    weight[0]=-3.0;
    weight[1]=-1.0;
    weight[2]=0.0;
    weight[3]=1.0;
    weight[4]=3.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* ***** OPERATING TEMP ***** */
/* ***** */

int temp_range(float temp)
{

```



```

int loc;

/* Definition of Ranges of "operating temp" for each situation */

/* TEMP RANGES SETUP */
/* -----*/

float ranges[][2]=
{{154.0,157.0}, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
 {158.0,161.0}, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
 {162.0,165.0}, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
 {166.0,169.0}, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
 {170.0,175.0}, // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
 {0.0,0.0},     // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
 {0.0,0.0}};   // Situation #6 (LOW_limit[][0],HIGH_limit[][1])

loc = rule_base_setA(temp,ranges);
return loc;
}

char *temp_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* TEMP STATUS SETUP */
    /* -----*/

    status[0]="LOW OPERATING TEMPERATURE\n"
              "-----";
    status[1]="LOW OPERATING TEMPERATURE\n"
              "-----";
    status[2]="NORMAL CONDITIONS\n"
              "-----";
    status[3]="HIGH OPERATING TEMPERATURE\n"
              "-----";
    status[4]="HIGH OPERATING TEMPERATURE\n"
              "-----";
    status[5]=" ";
    status[6]=" ";

    return status[loc];
}

char *temp_time_required(int loc)
{
    char *time_required[10];

    /* TEMP TIME_REQUIRED SETUP */
    /* -----*/

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="LONGER COOKING TIME\n"
                    "BY 15 TO 25 MINUTES";
    time_required[1]="LONGER COOKING TIME\n"
                    "BY 5 TO 15 MINUTES";
    time_required[2]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";

```

```

time_required[3]="SHORTER COOKING TIME\n"
               "BY 5 TO 15 MINUTES";

time_required[4]="SHORTER COOKING TIME\n"
               "BY 15 TO 25 MINUTES";

time_required[5]=" ";
time_required[6]=" ";

return time_required[loc];

}

char *temp_situation(int loc)
{
    char *situation[10];

    /* TEMP SITUATION SETUP */
    /* ----- */

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Good Pulp Brightness \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "      1- Steam Valve did not OPEN \n"
        "          on Time \n"
        "      2- Too much Condensate in the \n"
        "          Steam Valve \n"
        "      3- Poor Temperature Control \n"
        "      4- Poor Steam Quality \n"
        " ";

    situation[1]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Good Pulp Brightness \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "      1- Steam Valve did not OPEN \n"
        "          on Time \n"
        "      2- Too much Condensate in the \n"
        "          Steam Valve \n"
        "      3- Poor Temperature Control \n"
        "      4- Poor Steam Quality \n"
        " ";

    situation[2]=
        "          GOOD OPERATING TEMPERATURE \n"
        "          ----- \n"
        "          \n"

```

```

" RESULTS IN THE FINAL PULP:                \n"
"                                             \n"
"      1- Good Pulp Brightness                \n"
"                                             \n"
"      2- Good Kappa Number                  \n"
"                                             \n"
" ";

situation[3]=
"      UNDER THIS CONDITION                \n"
"      -----                            \n"
" RESULTS IN THE FINAL PULP:                \n"
"                                             \n"
"      1- Lower Pulp Brightness                \n"
"                                             \n"
" CAUSE:                                    \n"
"                                             \n"
"      1- Steam Valve did not CLOSE            \n"
"          on Time                            \n"
"      2- Too much Condensate in the            \n"
"          Steam Valve                        \n"
"      3- Poor Temperature Control            \n"
" ";

situation[4]=
"      UNDER THIS CONDITION                \n"
"      -----                            \n"
" RESULTS IN THE FINAL PULP:                \n"
"                                             \n"
"      1- Lower Pulp Brightness                \n"
"                                             \n"
" CAUSE:                                    \n"
"                                             \n"
"      1- Steam Valve did not CLOSE            \n"
"          on Time                            \n"
"      2- Too much Condensate in the            \n"
"          Steam Valve                        \n"
"      3- Poor Temperature Control            \n"
" ";

situation[5]=" ";
situation[6]=" ";

return situation[loc];

}

char *temp_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* TEMP TIME_LIMITS SETUP */
    /* -----*/

    // the TEMPERATURE high_time[0] and low_time[1] limits

    time_limits[0][0]="+25";
    time_limits[0][1]="+15";

    time_limits[1][0]="+15";
    time_limits[1][1]="+5";

```

```

time_limits[2][0]="GOOD";
time_limits[2][1]="GOOD";

time_limits[3][0]="-5";
time_limits[3][1]="-15";

time_limits[4][0]="-15";
time_limits[4][1]="-25";

time_limits[5][0]="GOOD";
time_limits[5][1]="GOOD";

time_limits[6][0]="GOOD";
time_limits[6][1]="GOOD";

return time_limits[loc][h1];
}

float temp_weight(int loc)
{
    float weight[10];

    /* TEMP WEIGHT SETUP */
    /* -----*/

    // the TEMP weight under specified condition

    weight[0]=4.0;
    weight[1]=2.0;
    weight[2]=0.0;
    weight[3]=-2.0;
    weight[4]=-4.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* *** OPERATING PRESSURE **** */
/* ***** */

int press_range(float temp)
{
    int loc;

    /* Definition of Ranges for "operating pressure" for each situation */

    /* PRESSURE RANGES SETUP */
    /* -----*/

    float ranges[][2]=
    {575.0,624.0, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
     625.0,674.0, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
     675.0,685.0, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
     686.0,735.0, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
     736.0,785.0, // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
     000.0,000.0, // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
     000.0,000.0}; // Situation #6 (LOW_limit[][0],HIGH_limit[][1])

    loc = rule_base_setA(temp,ranges);
    return loc;
}

```

```

char *press_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* PRESSURE TEMP STATUS SETUP */
    /* ----- */

    status[0]="LOW OPERATING PRESSURE\n"
              "-----";
    status[1]="LOW OPERATING PRESSURE\n"
              "-----";
    status[2]="NORMAL CONDITIONS\n"
              "-----";
    status[3]="HIGH OPERATING PRESSURE\n"
              "-----";
    status[4]="HIGH OPERATING PRESSURE\n"
              "-----";
    status[5]="";
    status[6]="";

    return status[loc];
}

char *press_time_required(int loc)
{
    char *time_required[10];

    /* PRESSURE TIME REQUIRED SETUP */
    /* ----- */

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="LONGER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES";
    time_required[1]="LONGER COOKING TIME\n"
                    "BY 0 TO 10 MINUTES";
    time_required[2]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";
    time_required[3]="SHORTER COOKING TIME\n"
                    "BY 0 TO 10 MINUTES";
    time_required[4]="SHORTER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES";
    time_required[5]=" ";
    time_required[6]=" ";

    return time_required[loc];
}

char *press_situation(int loc)
{
    char *situation[10];

```

```

/* PRESSURE SITUATION SETUP */
/* -----*/

// situation of the cooking behavior or results under this
// conditions. max(16 lines x 50 characters)

situation[0]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- Change in the SET POINT   \n"
"          \n"
"          2- Pressure RELIEF VALVE does \n"
"              not work properly        \n"
"          \n"
" ";

situation[1]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- Change in the SET POINT   \n"
"          \n"
"          2- Pressure RELIEF VALVE does \n"
"              not work properly        \n"
"          \n"
" ";

situation[2]=
"          GOOD OPERATING PRESSURE       \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Good Pulp Brightness      \n"
"          \n"
"          2- Good Kappa Number         \n"
"          \n"
" ";

situation[3]=
"          UNDER THIS CONDITION          \n"
"          -----                      \n"
"          \n"
" RESULTS IN THE FINAL PULP:           \n"
"          \n"
"          1- Lower Pulp Brightness     \n"
"          \n"
" CAUSE:                               \n"
"          \n"
"          1- Change in the SET POINT   \n"
"          \n"

```

```

        "
        "      2- Pressure RELIEF VALVE does      \n"
        "      not work properly                  \n"
        "                                          \n"
        " ";
situation[4]=
"      UNDER THIS CONDITION                      \n"
"      -----                                \n"
"      RESULTS IN THE FINAL PULP:                \n"
"      1- Lower Pulp Brightness                  \n"
"      CAUSE:                                    \n"
"      1- Change in the SET POINT                \n"
"      2- Pressure RELIEF VALVE does            \n"
"      not work properly                        \n"
" ";

situation[5]=" ";
situation[6]=" ";

return situation[loc];
}

char *press_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* PRESSURE TIME LIMITS SETUP */
    /* ----- */

    // the PRESSURE high_time[0] and low_time[1] limits

    time_limits[0][0]="+20";
    time_limits[0][1]="+10";

    time_limits[1][0]="+10";
    time_limits[1][1]="+0";

    time_limits[2][0]="GOOD";
    time_limits[2][1]="GOOD";

    time_limits[3][0]="-0";
    time_limits[3][1]="-10";

    time_limits[4][0]="-10";
    time_limits[4][1]="-20";

    time_limits[5][0]="GOOD";
    time_limits[5][1]="GOOD";

    time_limits[6][0]="GOOD";
    time_limits[6][1]="GOOD";

    return time_limits[loc][hl];
}

```



```

float press_weight(int loc)
{
    float weight[10];

    /* PRESSURE WEIGHT SETUP */
    /* -----*/

    weight[0]=3.0;
    weight[1]=1.0;
    weight[2]=0.0;
    weight[3]=-1.0;
    weight[4]=-3.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* **** TOTAL SO2 OF ACID **** */
/* ***** */

int total_range(float total)
{
    int loc;

    /* Definition of Ranges of "total SO2" for each situation */

    /* TOTAL SO2 RANGES SETUP */
    /* -----*/

    float ranges[][2]=
    {4.00,4.80, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
    4.81,5.44, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
    5.45,6.15, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
    6.16,6.50, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
    6.51,7.40, // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
    0.00,0.00, // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
    0.00,0.00}; // Situation #6 (LOW_limit[][0],HIGH_limit[][1])

    loc = rule_base_setA(total,ranges);
    return loc;
}

char *total_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* TOTAL SO2 STATUS SETUP */
    /* -----*/

    status[0]="LOW TOTAL SO2 OF ACID\n"
    "-----";
    status[1]="LOW TOTAL SO2 OF ACID\n"
    "-----";
    status[2]="NORMAL CONDITIONS\n"
    "-----";
    status[3]="HIGH TOTAL SO2 OF ACID\n"
    "-----";

```

```

status[4]="HIGH TOTAL SO2 OF ACID\n"
"-----";

status[5]=" ";
status[6]=" ";

return status[loc];
}

char *total_time_required(int loc)
{
    char *time_required[10];

    /* TOTAL SO2 TIME_REQUIRED SETUP */
    /* ----- */

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="LONGER COOKING TIME\n"
"BY 10 TO 20 MINUTES";

    time_required[1]="LONGER COOKING TIME\n"
"BY 0 TO 10 MINUTES";

    time_required[2]="WILL NOT CHANGE THE\n"
"NORMAL COOKING TIME";

    time_required[3]="SHORTER COOKING TIME\n"
"BY 0 TO 10 MINUTES";

    time_required[4]="SHORTER COOKING TIME\n"
"BY 10 TO 20 MINUTES";

    time_required[5]=" ";
    time_required[6]=" ";

    return time_required[loc];
}

char *total_situation(int loc)
{
    char *situation[10];

    /* TOTAL SO2 SITUATION SETUP */
    /* ----- */

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=
"      UNDER THIS CONDITION          \n"
"      -----                      \n"
"      \n"
"      RESULTS IN THE FINAL PULP:    \n"
"      \n"
"      1- Good Pulp Brightness       \n"
"      \n"
"      COOKING ACID QUALITY          \n"
"      \n"
"      1- POOR Quality               \n"

```

```

    " ";

situation[1]=
    "          UNDER THIS CONDITION          \n"
    "          -----                        \n"
    "          RESULTS IN THE FINAL PULP:      \n"
    "          1- Good Pulp Brightness         \n"
    "          COOKING ACID QUALITY             \n"
    "          1- POOR Quality                  \n"
    " ";

situation[2]=
    "          GOOD & TOTAL SO2 IN ACID        \n"
    "          -----                        \n"
    "          RESULTS IN THE FINAL PULP:      \n"
    "          1- Good Pulp Brightness         \n"
    "          2- Good Kappa Number             \n"
    " ";

situation[3]=
    "          UNDER THIS CONDITION          \n"
    "          -----                        \n"
    "          RESULTS IN THE FINAL PULP:      \n"
    "          1- Pulp Brightness could be lower \n"
    "          (check situation of FREE & COMBINE SO2) \n"
    "          COOKING ACID QUALITY             \n"
    "          1- Strong Cooking Acid           \n"
    " ";

situation[4]=
    "          UNDER THIS CONDITION          \n"
    "          -----                        \n"
    "          RESULTS IN THE FINAL PULP:      \n"
    "          1- Pulp Brightness could be lower \n"
    "          (check situation of FREE & COMBINE SO2) \n"
    "          COOKING ACID QUALITY             \n"
    "          1- Strong Cooking Acid           \n"
    " ";

situation[5]=" ";
situation[6]=" ";

return situation[loc];

}

char *total_time_limits(int loc,int hl)
{

```

```

    char *time_limits[10][2];

    /* TOTAL SO2 TIME_LIMITS SETUP */
    /* -----*/

    // the pH high_time[0] and low_time[1] limits

    time_limits[0][0]="+20";
    time_limits[0][1]="+10";

    time_limits[1][0]="+10";
    time_limits[1][1]="+0";

    time_limits[2][0]="GOOD";
    time_limits[2][1]="GOOD";

    time_limits[3][0]="-0";
    time_limits[3][1]="-10";

    time_limits[4][0]="-10";
    time_limits[4][1]="-20";

    time_limits[5][0]="GOOD";
    time_limits[5][1]="GOOD";

    time_limits[6][0]="GOOD";
    time_limits[6][1]="GOOD";

    return time_limits[loc][hl];
}

float total_weight(int loc)
{
    float weight[10];

    /* TOTAL SO2 WEIGHT SETUP */
    /* -----*/

    weight[0]=3.0;
    weight[1]=1.0;
    weight[2]=0.0;
    weight[3]=-1.0;
    weight[4]=-3.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* ***      CHIP LOAD      **** */
/* ***** */

int load_range(float load)
{
    int loc;

    /* Definition of Ranges of "chip load" for each situation */

    /* CHIP LOAD RANGES SETUP */
    /* -----*/

    float ranges[][2]=

```

```

    (22.0,29.0, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
    30.0,34.0, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
    35.0,40.0, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
    41.0,46.0, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
    47.0,55.0, // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
    0.0,0.0, // Situation #5 (LOW_limit[][0],HIGH_limit[][1])
    0.0,0.0); // Situation #6 (LOW_limit[][0],HIGH_limit[][1])

loc = rule_base_setA(load,ranges);
return loc;
}

char *load_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* CHIP LOAD STATUS SETUP */
    /* ----- */

    status[0]=" LOW CHIP LOAD \n"
              " ----- ";

    status[1]=" LOW CHIP LOAD \n"
              " ----- ";

    status[2]="NORMAL CONDITIONS\n"
              "-----";

    status[3]=" HEAVY CHIP LOAD \n"
              " ----- ";

    status[4]=" HEAVY CHIP LOAD \n"
              " ----- ";

    status[5]=" ";
    status[6]=" ";

    return status[loc];
}

char *load_time_required(int loc)
{
    char *time_required[10];

    /* CHIP LOAD TIME_REQUIRED SETUP */
    /* ----- */

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="SHORTER COOKING TIME\n"
                    "BY 15 TO 25 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[1]="SHORTER COOKING TIME\n"
                    "BY 5 TO 15 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[2]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";

```

```

time_required[3]="LONGER COOKING TIME\n"
                "BY 5 TO 15 MINUTES\n"
                "[low accuracy of S-factor]";

time_required[4]="LONGER COOKING TIME\n"
                "BY 15 TO 25 MINUTES\n"
                "[low accuracy of S-factor]";

time_required[5]=" ";
time_required[6]=" ";

return time_required[loc];

}

char *load_situation(int loc)
{
    char *situation[10];

    /* CHIP LOAD SITUATION SETUP */
    /* ----- */

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "          1- Lower Pulp Yield \n"
        "          2- Higher Pulp Brightness \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "          1- Chips are not well Packed \n"
        "          2- Steam Packer did not work \n"
        "          properly";

    situation[1]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "          1- Lower Pulp Yield \n"
        "          2- Higher Pulp Brightness \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "          1- Chips are not well Packed \n"
        "          2- Steam Packer did not work \n"
        "          properly";

    situation[2]=
        "          GOOD CHIP LOAD VALUE          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "          1- Good Pulp Brightness \n"
        "          \n"
        "          2- Good Kappa Number \n"

```

```

"                                     \n"
" ";
situation[3]=
"          UNDER THIS CONDITION          \n"
"          ----- \n"
"          RESULTS IN THE FINAL PULP: \n"
"          \n"
"          1- Lower Pulp Brightness \n"
"          \n"
"          RESULTS DURING OPERATION: \n"
"          \n"
"          1- Reduction of Acid Strength \n"
"          2- Higher Steam Demand \n"
"          \n"
"          CAUSE: \n"
"          \n"
"          1- Higher Moisture Content in \n"
"              Wood Chips";

situation[4]=
"          UNDER THIS CONDITION          \n"
"          ----- \n"
"          RESULTS IN THE FINAL PULP: \n"
"          \n"
"          1- Lower Pulp Brightness \n"
"          \n"
"          RESULTS DURING OPERATION: \n"
"          \n"
"          1- Reduction of Acid Strength \n"
"          2- Higher Steam Demand \n"
"          \n"
"          CAUSE: \n"
"          \n"
"          1- Higher Moisture Content in \n"
"              Wood Chips";

situation[5]=" ";
situation[6]=" ";

return situation[loc];

}

char *load_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* CHIP LOAD TIME_LIMITS SETUP */
    /* -----*/

    // the pH high_time[0] and low_time[1] limits

    time_limits[0][0]="-15";
    time_limits[0][1]="-25";

    time_limits[1][0]="-5";
    time_limits[1][1]="-15";

    time_limits[2][0]="GOOD";
    time_limits[2][1]="GOOD";

```



```

time_limits[3][0]="+15";
time_limits[3][1]="+5";

time_limits[4][0]="+25";
time_limits[4][1]="+15";

time_limits[5][0]="GOOD";
time_limits[5][1]="GOOD";

time_limits[6][0]="GOOD";
time_limits[6][1]="GOOD";

return time_limits[loc][h1];

)

float load_weight(int loc)
{
    float weight[10];

    /* CHIP LOAD WEIGHT SETUP */
    /* -----*/

    weight[0]=-4.0;
    weight[1]=-2.0;
    weight[2]=0.0;
    weight[3]=2.0;
    weight[4]=4.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* ***      CHIP SIZE      **** */
/* ***** */

int size_range(float size)
{
    int loc;

    /* Definition of Ranges of "chip size" for each situation */

    /* CHIP LOAD RANGES SETUP */
    /* -----*/

    float ranges[]=
        {0.0, // Situation #0
         1.0, // Situation #1
         2.0, // Situation #2
         3.0, // Situation #3
         0.0, // Situation #4
         0.0, // Situation #5
         0.0}; // Situation #6

    loc = rule_base_setB(size,ranges);
    return loc;
}

char *size_status(int loc)
{
    char *status[10];

```

```

/* CHIP SIZE STATUS SETUP */
/* -----*/

status[0]="";

status[1]="NORMAL CONDITIONS\n"
        "-----";

status[2]="SMALL SIZE OF CHIPS\n"
        "-----";

status[3]="LARGE SIZE OF CHIPS\n"
        "-----";


status[4]=" ";
status[5]=" ";
status[6]=" ";

return status[loc];

}

char *size_time_required(int loc)
{
    char *time_required[10];

    /* CHIP SIZE TIME_REQUIRED SETUP */
    /* -----*/

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]=" ";

    time_required[1]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";

    time_required[2]="SHORTER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[3]="LONGER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[4]=" ";
    time_required[5]=" ";
    time_required[6]=" ";

    return time_required[loc];

}

char *size_situation(int loc)
{
    char *situation[10];

    /* CHIP SIZE SITUATION SETUP */
    /* -----*/

```

```

// situation of the cooking behavior or results under this
// conditions. max(16 lines x 50 characters)

situation[0]=" ";

situation[1]=
"          GOOD CHIP LOAD VALUE          \n"
"          ----- \n"
"          \n"
" RESULTS IN THE FINAL PULP: \n"
"          \n"
"          1- Good Pulp Brightness \n"
"          \n"
"          2- Good Kappa Number \n"
"          \n"
" ";

situation[2]=
"          UNDER THIS CONDITION          \n"
"          ----- \n"
"          \n"
" RESULTS IN THE FINAL PULP: \n"
"          \n"
"          1- Lower Pulp Brightness \n"
"          \n"
" RESULTS DURING OPERATION: \n"
"          \n"
"          1- Lower Steam Demand \n"
"          \n"
" CAUSE: \n"
"          \n"
"          1- Chip Size is too SMALL \n"
" ";

situation[3]=
"          UNDER THIS CONDITION          \n"
"          ----- \n"
"          \n"
" RESULTS DURING OPERATION: \n"
"          \n"
"          1- Difficult to Cook the Chips \n"
"          2- Difficult for the cooking acid \n"
"             to penetrate into the chips \n"
"          \n"
" CAUSE: \n"
"          \n"
"          1- Chip Size is too LARGE";

situation[4]=" ";
situation[5]=" ";
situation[6]=" ";

return situation[loc];

}

char *size_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* CHIP LOAD TIME_LIMITS SETUP */
    /* -----*/

    // the SIZE high_time[0] and low_time[1] limits

```

```

time_limits[0][0]="GOOD";
time_limits[0][1]="GOOD";

time_limits[1][0]="GOOD";
time_limits[1][1]="GOOD";

time_limits[2][0]="-10";
time_limits[2][1]="-20";

time_limits[3][0]="+20";
time_limits[3][1]="+10";

time_limits[4][0]="GOOD";
time_limits[4][1]="GOOD";

time_limits[5][0]="GOOD";
time_limits[5][1]="GOOD";

time_limits[6][0]="GOOD";
time_limits[6][1]="GOOD";

return time_limits[loc][hl];
}

float size_weight(int loc)
{
    float weight[10];

    /* CHIP SIZE WEIGHT SETUP */
    /* ----- */

    weight[0]=0.0;
    weight[1]=0.0;
    weight[2]=-3.0;
    weight[3]=3.0;
    weight[4]=0.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* ***      CHIP QUALITY      *** */
/* ***** */

int quality_range(float quality)
{
    int loc;

    /* Definition of Ranges of "operating pressure" for each situation */

    /* CHIP QUALITY RANGES SETUP */
    /* ----- */

    float ranges[]=
        (0.0, // Situation #0
         1.0, // Situation #1
         2.0, // Situation #2
         3.0, // Situation #3
         4.0, // Situation #4

```

```

        0.0, // Situation #5
        0.0); // Situation #6

loc = rule_base_setB(quality,ranges);
return loc;
}

char *quality_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* CHIP QUALITY STATUS SETUP */
    /* -----*/

    status[0]="";

    status[1]="NORMAL CONDITIONS\n"
              "-----";

    status[2]="POOR CHIP QUALITY\n"
              "-----";

    status[3]="HIGH % OF HARDWOOD\n"
              "-----";

    status[4]="POOR CHIP QUALITY\n"
              "-----";

    status[5]=" ";
    status[6]=" ";

    return status[loc];
}

char *quality_time_required(int loc)
{
    char *time_required[10];

    /* CHIP QUALITY TIME_REQUIRED SETUP */
    /* -----*/

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]=" ";

    time_required[1]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";

    time_required[2]="SHORTER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[3]="SHORTER COOKING TIME\n"
                    "BY 10 TO 20 MINUTES\n"
                    "[low accuracy of S-factor]";

    time_required[4]="MUCH SHORTER COOKING TIME\n"
                    "BY ??? MINUTES\n"
                    "[low accuracy of S-factor]";

```

```

time_required[5]=" ";
time_required[6]=" ";

return time_required[loc];

}

char *quality_situation(int loc)
{
    char *situation[10];

    /* CHIP QUALITY SITUATION SETUP */
    /* -----*/

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=

    situation[1]=
        "          GOOD CHIP QUALITY          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Good Pulp Brightness \n"
        "          \n"
        "      2- Good Kappa Number \n"
        "          \n"
        " ";

    situation[2]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Lower Pulp Brightness \n"
        "          \n"
        " RESULTS DURING OPERATION: \n"
        "          \n"
        "      1- Lower Steam Demand \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "      1- POOR Chip Quality";

    situation[3]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Higher Pulp Brightness \n"
        "      2- Lower Pulp Strenght \n"
        "          \n"
        " RESULTS DURING OPERATION: \n"
        "          \n"
        "      1- Could Cause Problem at the \n"
        "          Washing & Screening and Bleachery \n"
        "          \n"
        " CAUSE: \n"
        "          \n"
        "      1- High Hardwood Content in the Digester";

```

```

situation[4]=
    "          UNDER THIS CONDITION          \n"
    "          ----- \n"
    "          RESULTS IN THE FINAL PULP: \n"
    "          1- Pulp Brightness ?????? \n"
    "          2- Lower Pulp Strenght \n"
    "          RESULTS DURING OPERATION: \n"
    "          1- Lower Steam Demand \n"
    "          2- Could Cause Problem at the \n"
    "             Washing & Screening and Bleachery \n"
    "          CAUSE: \n"
    "          1- POOR Chip Quality";

situation[5]=" ";
situation[6]=" ";

return situation[loc];

}

char *quality_time_limits(int loc,int hl)
{
    char *time_limits[10][2];

    /* CHIP QUALITY TIME_LIMITS SETUP */
    /* ----- */

    // the chip quality low_time[0] and high_time[1] limits

    time_limits[0][0]="GOOD";
    time_limits[0][1]="GOOD";

    time_limits[1][0]="GOOD";
    time_limits[1][1]="GOOD";

    time_limits[2][0]="-10";
    time_limits[2][1]="-20";

    time_limits[3][0]="-10";
    time_limits[3][1]="-20";

    time_limits[4][0]="-10";
    time_limits[4][1]="-35";

    time_limits[5][0]="GOOD";
    time_limits[5][1]="GOOD";

    time_limits[6][0]="GOOD";
    time_limits[6][1]="GOOD";

    return time_limits[loc][hl];

}

float quality_weight(int loc)

```



```

{
    float weight[10];

    /* CHIP QUALITY WEIGHT SETUP */
    /* ----- */

    weight[0]=0.0;
    weight[1]=0.0;
    weight[2]=-3.0;
    weight[3]=-3.0;
    weight[4]=-30.0;
    weight[5]=0.0;
    weight[6]=0.0;

    return weight[loc];
}

/* ***** */
/* **** FREE & COMBINE SO2 OF ACID **** */
/* ***** */

int freecom_range(float free, float combine)
{
    int loc;
    float def;

    /* Definition of Ranges of "operating pressure" for each situation */

    /* TOTAL SO2 RANGES SETUP */
    /* ----- */

    float ranges[][2]=
    {-2.00,0.15, // Situation #0 (LOW_limit[][0],HIGH_limit[][1])
     0.15,0.20, // Situation #1 (LOW_limit[][0],HIGH_limit[][1])
     0.20,2.00, // Situation #2 (LOW_limit[][0],HIGH_limit[][1])
     0.00,0.00, // Situation #3 (LOW_limit[][0],HIGH_limit[][1])
     0.00,0.00}; // Situation #4 (LOW_limit[][0],HIGH_limit[][1])
    def=free-combine;
    loc = rule_base_setA(def,ranges);
    return loc;
}

char *freecom_status(int loc)
{
    char *status[10];

    // high, low, or normal value. max(2 lines x 30 characters)

    /* FREE & COMBINE SO2 STATUS SETUP */
    /* ----- */

    status[0]="LOW FREE or HIGH COMBINE\n"
             "-----";

    status[1]="NORMAL CONDITIONS\n"
             "-----";

    status[2]="HIGH FREE or LOW COMBINE\n"
             "-----";

    status[3]=" ";
    status[4]=" ";
    status[5]=" ";
    status[6]=" ";

```

```

return status[loc];
}

char *freecom_time_required(int loc)
{
    char *time_required[10];

    /* FREE & COMBINE SO2 TIME_REQUIRED SETUP */
    /* ----- */

    // time required under this condition max(3 lines x 30 characters)

    time_required[0]="CHECK pH AND TOTAL SO2\n"
                    "OF ACID FOR COOKING TIME\n"
                    "REQUIRED";

    time_required[1]="WILL NOT CHANGE THE\n"
                    "NORMAL COOKING TIME";

    time_required[2]="CHECK pH AND TOTAL SO2\n"
                    "OF ACID FOR COOKING TIME\n"
                    "REQUIRED";

    time_required[3]=" ";
    time_required[4]=" ";
    time_required[5]=" ";
    time_required[6]=" ";

    return time_required[loc];
}

char *freecom_situation(int loc)
{
    char *situation[10];

    /* FREE & COMBINE SO2 SITUATION SETUP */
    /* ----- */

    // situation of the cooking behavior or results under this
    // conditions. max(16 lines x 50 characters)

    situation[0]=
        "          UNDER THIS CONDITION          \n"
        "          ----- \n"
        "          \n"
        " RESULTS IN THE FINAL PULP: \n"
        "          \n"
        "      1- Good Pulp Brightness \n"
        "          \n"
        " COOKING ACID QUALITY: \n"
        "          \n"
        "      1- Could Increase pH of Cooking Acid \n"
        "      2- Lower Acidity of Cooking Acid \n"
        " *, \n";

    situation[1]=
        "      GOOD FREE & COMBINE SO2 IN ACID \n"
        "      ----- \n"
        " \n";

```

```

" RESULTS IN THE FINAL PULP:                                \n"
"                                                             \n"
"      1- Good Pulp Brightness                               \n"
"                                                             \n"
"      2- Good Kappa Number                                   \n"
"                                                             \n"
" ";

situation[2]=
"      UNDER THIS CONDITION                                \n"
"      -----                                              \n"
"      RESULTS IN THE FINAL PULP:                            \n"
"                                                             \n"
"      1- Lower Pulp Brightness                              \n"
"                                                             \n"
"      COOKING ACID QUALITY:                                  \n"
"                                                             \n"
"      1- Could Decrease pH of ACID                          \n"
"      2- Higher Acidity of Cooking Acid                     \n"
" ";

situation[3]=" ";
situation[4]=" ";
situation[5]=" ";
situation[6]=" ";

return situation[loc];
}

/* ***** */
/* **** COUPLED OPERATING TEMP. AND OPERATING PRESS. **** */
/* ***** */

void press_temp_couple(float press, float temp)
{
int loct, locp;

loct = temp_range(temp);
locp = press_range(press);

if(temp_weight(loct) == 0.0){
couple_time_limit[2][0] = press_time_limits(locp,0);
couple_time_limit[2][1] = press_time_limits(locp,1);
couple_weight[2] = press_weight(locp);
return;
}

if(press_weight(locp) == 0.0){
couple_time_limit[2][0] = temp_time_limits(loct,0);
couple_time_limit[2][1] = temp_time_limits(loct,1);
couple_weight[2] = temp_weight(loct);
return;
}

couple_weight[2] = (temp_weight(loct)+press_weight(locp));
weight_table(couple_weight[2]);
couple_time_limit[2][0] = ul;
couple_time_limit[2][1] = ll;
return;
}

/* ***** */

```

```

/* **** COUPLED TOTAL SO2 & pH ---> COOKING ACID **** */
/* **** */

void ph_total_couple(float ph, float total)
{
    int loct, locp;

    loct = total_range(total);
    locp = ph_range(ph);

    if(total_weight(loct) == 0.0){
        couple_time_limit[1][0] = ph_time_limits(locp,0);
        couple_time_limit[1][1] = ph_time_limits(locp,1);
        couple_weight[1] = ph_weight(locp);
        return;
    }

    if(ph_weight(locp) == 0.0){
        couple_time_limit[1][0] = total_time_limits(loct,0);
        couple_time_limit[1][1] = total_time_limits(loct,1);
        couple_weight[1] = total_weight(loct);
        return;
    }

    couple_weight[1] = (total_weight(loct)+ph_weight(locp));
    weight_table(couple_weight[1]);
    couple_time_limit[1][0] = ul;
    couple_time_limit[1][1] = ll;
    return;

}

/* **** */
/* **** COUPLED CHIP LOAD-QUALITY-SIZE --> WOOD CHIPS **** */
/* **** */

void load_quali_size_couple(float load, float quality, float size)
{
    int locl, locq, locs;

    locl = load_range(load);
    locq = quality_range(quality);
    locs = size_range(size);

    //if(load_weight(locl)==0.0 && quality_weight(locq)==0.0){
    //    couple_time_limit[0][0] = size_time_limits(locs,0);
    //    couple_time_limit[0][1] = size_time_limits(locs,1);
    //    couple_weight[0] = size_weight(locs);
    //    return;
    //}

    //if(load_weight(locl)==0.0 && size_weight(locs)==0.0){
    //    couple_time_limit[0][0] = quality_time_limits(locq,0);
    //    couple_time_limit[0][1] = quality_time_limits(locq,1);
    //    couple_weight[0] = quality_weight(locq);
    //    return;
    //}

    //if(size_weight(locs)==0.0 && quality_weight(locq)==0.0){
    //    couple_time_limit[0][0] = load_time_limits(locl,0);
    //    couple_time_limit[0][1] = load_time_limits(locl,1);
    //    couple_weight[0] = load_weight(locl);
    //    return;
    //}

    couple_weight[0] =
    (load_weight(locl)+quality_weight(locq)+size_weight(locs));
    weight_table(couple_weight[0]);

```

```

couple_time_limit[0][0] = ul;
couple_time_limit[0][1] = ll;
return;

}

/* ***** */
/* ****    COUPLE CHIPS - COOKING ACID - TEMP&PRESS    **** */
/* ***** */

void all_couple()
{
    //if(couple_weight[0]==0.0 && couple_weight[1]==0.0){
    //    couple_time_limit[3][0] = couple_time_limit[2][0];
    //    couple_time_limit[3][1] = couple_time_limit[2][1];
    //    return;
    // }
    //if(couple_weight[0]==0.0 && couple_weight[2]==0.0){
    //    couple_time_limit[3][0] = couple_time_limit[1][0];
    //    couple_time_limit[3][1] = couple_time_limit[1][1];
    //    return;
    // }
    //if(couple_weight[1]==0.0 && couple_weight[2]==0.0){
    //    couple_time_limit[3][0] = couple_time_limit[0][0];
    //    couple_time_limit[3][1] = couple_time_limit[0][1];
    //    return;
    // }
    couple_weight[3] = (couple_weight[0]+couple_weight[1]+couple_weight[2]);
    weight_table(couple_weight[3]);
    couple_time_limit[3][0] = ul;
    couple_time_limit[3][1] = ll;
    return;
}

void weight_table(float x)
{
    if( x >= 10.0 ){ul="+55";
                    ll="+45";}
        if( x==9.0 ){ ul="+50";
                       ll="+40";}
        if( x==8.0 ){ ul="+45";
                       ll="+35";}
        if( x==7.0 ){ ul="+40";
                       ll="+30";}
        if( x==6.0 ){ ul="+35";
                       ll="+25";}
        if( x==5.0 ){ ul="+30";
                       ll="+20";}
        if( x==4.0 ){ ul="+25";
                       ll="+15";}
        if( x==3.0 ){ ul="+20";
                       ll="+10";}
        if( x==2.0 ){ ul="+15";
                       ll="+5";}
        if( x==1.0 ){ ul="+10";
                       ll="+0";}
        if( x==0.0 ){ ul="GOOD";
                       ll="GOOD";}
        if( x== -1.0 ){ ul="-0";
                        ll="-10";}
        if( x== -2.0 ){ ul="-5";
                        ll="-15";}
}

```

```
    if( x== -3.0 ){ ul="-10";  
                    ll="-20";}  
    if( x== -4.0 ){ ul="-15";  
                    ll="-25";}  
    if( x== -5.0 ){ ul="-20";  
                    ll="-30";}  
    if( x== -6.0 ){ ul="-25";  
                    ll="-35";}  
    if( x== -7.0 ){ ul="-30";  
                    ll="-40";}  
    if( x== -8.0 ){ ul="-35";  
                    ll="-45";}  
    if( x== -9.0 ){ ul="-40";  
                    ll="-50";}  
    if( x<= -10.0 ){ ul="-45";  
                    ll="-55";}  
}
```

```

// *****
//
//          MODULE: "MAINPRO.C"
//
//
// *****

#define MAIN 1
#include <stdarg.h>
#include <window.h>
#include <process.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <string.h>
#include <alloc.h>
#include <mem.h>

/* ***** */
void far movescreenmem(char far *source, char far *dest, unsigned len,
int snowcheck)
{
    unsigned i;
    i = snowcheck; /* dummy line to remove warning */
    for (i = 0; i < len; i++)
        *(dest+i) = *(source+i);
}
/* ***** */

void popmenu();
void start_up();
void popmenu_option();
void init_screen();
void intelligent(char index);
void situations(char index);
void equip_spec(char index);
void rw_stdcond();
void set_stdcond(void);
void std_bar(char index);
void shut_down();
void pulping_inf();
void introduc();
void ggrraph();
int godomenu();
float windfunc();
float wind2unc();
float database[10];
char numfunc();
extern float s_fact[4];

main()
{
    system("cover.exe");
    initwindow();
    shadow=FALSE;
    plcolor =attr(15,0);
    pletcolor=attr(12,black);
    pbrdcolor=attr(15,4);
    plhicolor=attr(red,white);
    windowcolor = attr(0,0);

```



```

bordercolor = attr(1,1);
init_screen();
set_stdcond();
rw_stdcond( "stdcond.knb",0);
s_fact[0]=0.0;
godomenu();
setcursor(on);
screenmode(color80);
clrscr(attr(white,black),black);
exit(1);
return;
}

godomenu()
{
    char *menu1, *first;
    char *readstuff(char *);
    char index, index2, *ky;
    int see;
    first=readstuff("first.txx");
    mwind(5,2,73,20,attr(1,1),attr(14,1),1);
    wtitle(CENTER,TOP,attr(14,1)," ATTENTION !!! ");
    wtitle(CENTER,BOTTOM,attr(14,1)," press << ANY KEY >> to
continue ");
    pwindf(11,1,attr(15,1),"%s",first);
    see = keyin();
    rwind();
    free(first);

    pullmainmenu[0]=
    "inTro\n"
    "Pulping\n"
    " speC\n"
    "Inputs\n"
    "Situations\n"
    "Option\n"
    "Next menu\n"
    "TPCISON\n";

    pullsubmenu[0][2]=
    "Digesters      \n"
    "Acid system    \n"
    "Condensate system\n"
    "Water system   \n"
    "dRain down system\n"
    "dumP system    \n"
    "Flushing liquor \n"
    "DACWRPF\n";

    pullsubmenu[0][3]=
    "chip Load      \n"
    "chip Quality    \n"
    "chip Size       \n"
    "Total % SO2     \n"
    "Combine % SO2   \n"
    "Free % SO2      \n"
    "pH              \n"
    "opera. teMperature\n"
    "opera. Pressure  \n"
    "s-factor moDel  \n"
    "side Relief     \n"
    "LQSTCFHMPDR\n";

    pullsubmenu[0][4]=

```

```

"chip Load          \n"
"chip Quality        \n"
"chip size           \n"
"Total % SO2         \n"
"Free & Combine      \n"
"pH                  \n"
"opera. teMperature \n"
"opera. Pressure     \n"
"Cooking Time        \n"
"Time - Kappa        \n"
"LQZTFHMPCK\n";

    pullmainmenu[1]=
"Prev. menu\n"
"Operation\n"
"Flow Dia\n"
"Shutdown\n"
"startUp\n"
"Editor\n"
"POFSUE\n";

    while((index=pullmenu(top,0)) !=NULL)
    {
        switch(pullmenuloc)
        {
            case 0:
                introduc();
                break;

            case 1:
                pulping_inf();
                break;

            case 2:
                shadow=TRUE;
                equip_spec(index);
                shadow=FALSE;
                break;

            case 3:
                intelligent(index);
                break;

            case 4:
                situations(index);
                break;

            case 5:
                popmenu_option();
                break;

            case 6:
                while((index=pullmenu(top,1)) !=NULL)
                {
                    switch(pullmenuloc)
                    {
                        case 0:
                            index = NULL;
                            break;

                        case 1:

```

```

        opera_steps();
        break;

        case 2:
            rw_stdcond( "currcond.dat",1);
            rwind();
            rwind();
            rwind();
            system("flow.exe");
            init_screen();
            break;

        case 3:
            shut_down();
            break;

        case 4:
            start_up();
            break;

        case 5:
            popmenu();
            init_screen();
            break;
    }
    if(index==NULL) break;
}
}
return;
}

void init_screen()
{
    screenmode(color80);
    clrscr(attr(white,9),9);
    setcursor(off);
    mwind(78,0,80,25,attr(0,0),attr(0,0),5);
    mwind(1,24,77,25,attr(7,7),attr(7,7),5);
    wtitle(CENTER,TOP,attr(4,7),"INTELLIGENT OPERATION SUPPORT
SYSTEM\* FOR THE BATCH DIGESTERS");
    mwind(0,0,77,0,attr(0,0),attr(0,0),5);
}

void popmenu()
{
    char index2, *menu1;

    menu1 =
        "\n"
        "PE2 \n"
        "P\n";

    index2=popupmenu(60,1,menu1,2);
    if(index2=='P'){
        rwind();
        rwind();
        rwind();
        rwind();
        spawnl(P_WAIT,"pe2.exe",NULL);
        return;
    }
    if(index2==NULL){ rwind();
        rwind();

```

```

        rwind();
        rwind();
        return;
    }

void popmenu_option()
{
    char index2, *menu1;

    menu1 =
        "\n"
        "Bar graph          \n"
        "bar graph s-Factor \n"
        "Current conditions \n"
        "Standard conditions\n"
        "BFCS\n";

    index2=popupmenu(60,1,menu1,2);
    std_bar(index2);
    rwind();
    return;
}

```

```

// *****
//
//          MODULE: "MMODEL.C"
//
//
// *****

# include <stdarg.h>
# include <window.h>
# include <process.h>
# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>
extern float database[10];
extern char shadow;
float get_valu(int, int, int, char *, float, float);
float s_fact_model(float);
send_data(float *cinputs, int imax, char *filename);
float change_to_min(float);
float s_fact[4];
void enter_key();
float windfunc(char *, char *, char *, char *, int, int, int);

/*   C O O K I N G   M O D E L   */

cooking_model()
{
    float sfact,tm,hr;
    float cmodelp[3];
    int maxdig,see;
    char *tSO20,*tSO210,*kappa,*s,*sfactor,*sfact10,*tme;
    char *readstuff(char *);

    tSO210=
        "Enter the % total SO2 of the cooking acid sample\n"
        "collected 10 minutes after the reaction temperature\n"
        "has reached a MAXIMUM .....[x.xx] -->";
    sfactor=
        "The S-FACTOR required is ..... -->";
    sfact10=
        "Enter the S-factor at that time .[xx.xx].or.[xxx.x] -->";
    tme=
        "Enter the TIME the sample was collected [hr.min]\n"
        "(i.e. time from the start of the cycle) .....[x.xx] -->";
    sfact = 0.0;
    mwind(5,1,73,19,attr(4,4),attr(8,7),3);
    wtitle(CENTER,TOP,attr(0,7)," COOKING MODEL ");

    pwindf(1,1,attr(15,4),"%s",tSO210);
    maxdig = 3;
    cmodelp[1] = get_valu(56,3,maxdig,"%1.2f",1.0,3.5);
    if(cmodelp[1]==999.0) return;
    database[9]=cmodelp[1];

    pwindf(1,5,attr(15,4),"%s",tme);
    s_fact[2] = get_valu(56,6,maxdig,"%1.2f",1.50,3.50);
    if(s_fact[2]==999.0) return;
    s_fact[2]=change_to_min(s_fact[2]);

```

```

pwindf(58,6,attr(0,7),":");

pwindf(1,8,attr(15,4),"%s",sfact10);
maxdig = 4;
s_fact[1]= get_valu(56,8,maxdig,"%3.1f",50.0,250.0);
if(s_fact[1]==999.0) return;

shadow=TRUE;
mwind(0,9,24,17,attr(7,7),attr(14,7),5);
    pwindfc(2,attr(0,7),"%s"," Please Wait ... ");
setcursor(ON);
gowindcr(23,3);
cmodelp[0]=database[3];
cmodelp[2]=28.0;
send_data(cmodelp,2,"SFACTINP.DAT");
see = spawnl(P_WAIT,"cookmodl.exe",NULL);
if(see==-1) pwindfc(2,attr(14,4),"%s"," Execution Failed !!! ");
s = readstuff("SFACTOUT.DAT");
setcursor(OFF);
wtitle(CENTER,BOTTOM,attr(0,7),"press <ESC> to quit");
escape_key();
rwind();
shadow=FALSE;
sfact = atof(s);
if(see==-1) sfact=00.00;
s_fact[0]=sfact;
wtitle(CENTER,BOTTOM,attr(0,7)," press <ESC> to continue ");
mwind(10,13,68,17,attr(10,10),attr(14,10),2);
pwindf(3,1,attr(0,10),"%s",sfactor);
pwindf(47,1,attr(15,0),"%8.2f",sfact);
escape_key();
rwind();
rwind();
return;
}

float get_valu(int px, int py, int e,char *fof, float f, float g)
{
    char chl[5];
    int i, x2;
    float valu;
    valu=0.0;
    x2=0;
    while(valu < f || valu > g){
        chl[0]='.';
        chl[1]='.';
        chl[2]='.';
        chl[3]='.';
        chl[4]='.';
        setcursor(ON);
        gowindcr(px,py);
        pwindf(px,py,attr(0,7),"          ");
        for(i=0; i<=e; i++){
            for(;;){
                x2 = keyin();
                if(x2 == ESC) {
                    setcursor(OFF);
                    rwind();
                    return 999.0;
                }
                if(x2 == BS && i >= 1)
                {
                    pwindf((px-1)+i,py,attr(0,7)," ");
                    gowindcr(px-1+i,py);
                }
            }
        }
    }
}

```

```

        i--;
    }

    if(x2 >= 48 && x2 <= 57) break;
    if(x2==46) break;
    if(x2==CR){i=e+2;
        break;}

    if(x2!=8) sound_func(4000);

    }
    if(x2!=CR){
        x2 = x2-48;
        chl[i] = numfunc(x2);
        pwindf(px+i,py,attr(0,7),"%c",chl[i]);
        gowindcr(px+1+i,py);
    }
    }
    valu = atof(chl);
    setcursor(OFF);
    pwindf(px,py,attr(15,7),"      ");
    pwindf(px+1,py,attr(0,7),fof,valu);
}
return valu;
}

send_data(float *cinputs,int imax,char *filename)
{
    FILE *fp;
    register int i;

    if ( (fp=fopen(filename,"w"))==NULL){
        printf("cannot open directory file\n");
        exit(-1);
    }

    for(i=0; i<=imax; i++){
        fprintf(fp,"%1.2f\n",cinputs[i]);
    }
    fclose(fp);
    return;
}

float s_fact_model(float sf)
{
    double a,b,c,d,ka;

    a = 108.159;
    b = 0.0217;
    c = 8.336;
    d = 7.425;

    ka = (a - b*sf - c*database[3] - d*database[9]);
    return ka;
}

/* ----- S I D E   R E L I E F -----
*/

relief_func()
{
    char *select, *unit, *titl, *ques;
    float ca;
    double y,tm,sr;
    int maxdig,minl,maxl;

```



```

{
  select=
    " ENTER VOLUME (m3) -->";

  unit=
    " m3 ";

  titl=
    " S I D E R E L I E F ";

  ques=
    "What is the total volume of COOKING\n"
    " ACID entered in the digester ";

    maxdig = 2;
    minl = 125;
    maxl = 200;

  ca = windfunc(titl,ques,select,unit,maxdig,minl,maxl);
  if(ca == 999) return;

  y = 2.692169*database[0] - 18.840078;
  y = y*0.48*1.1*2000;
  tm = 0.18*y*100.0;
  tm = tm/(database[3]*2088.668);
  mwind(1,14,75,22,attr(15,6),attr(14,7),2);
  pwindf(12,1,attr(15,6),"Minimum Cooking Acid Required ----->");
  pwindf(12,2,attr(15,6),"Calculated Cooking Acid Required --->");
  pwindf(53,1,attr(0,7)," ");
  pwindf(53,2,attr(0,7)," ");
  pwindf(54,1,attr(0,7),"%.1f",125.0);
  pwindf(59,1,attr(0,7),"s",unit);
  pwindf(54,2,attr(0,7),"%.1f",tm);
  pwindf(59,2,attr(0,7),"s",unit);
  pwindfc(4,attr(15,0),"
");
  pwindfc(5,attr(15,0),"
");
  pwindfc(6,attr(15,0),"
");
  pwindf(18,5,attr(14,0),"Volume of SIDE RELIEF --->");
  if(tm <= 125.0) tm=125.0;
  sr=ca-tm;
  pwindf(46,5,attr(4,7)," ");
  pwindf(47,5,attr(4,7),"%.1f",sr);
  pwindf(51,5,attr(4,7),"s",unit);
  wtitle(CENTER,BOTTOM,attr(14,7),"press <ESC> to quit");
  escape_key();
  rwind();
  rwind();
}
return;
}

```

```

// *****
//
//          MODULE: "OPTIONS.C"
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

float get_valu(int px, int py, int e, char *fof, float f, float g);
float reaction_rate();
void curent_func();
void bar_graph1();
void bar_graph2();
void bar_graph3();

int bar_pos(char *st);
int choose_color(int pos);
void standard_func();
void make_graphw(int);
void bar_func();
void make_windl(char *title, float database[10],
                int a1, int a2, int a3, int a4, int a5, int a6);
void make_bars(int x1, int x2, int yt, int ytt);
extern char shadow;
extern float database[10];
extern float s_fact[4];
extern char *couple_time_limit[4][2];
extern float couple_weight[3];
int temp_range(float);
int press_range(float);
int load_range(float);
int ph_range(float);
int total_range(float);
int size_range(float);
int quality_range(float);
char *temp_time_limits(int, int);
char *load_time_limits(int, int);
char *ph_time_limits(int, int);
char *press_time_limits(int, int);
char *total_time_limits(int, int);
char *quality_time_limits(int, int);
char *size_time_limits(int, int);
char *quality_selection(float);
char *size_selection(float);

void press_temp_couple(float, float);
void ph_total_couple(float, float);
void load_quali_size_couple(float, float, float);
void all_couple(void);
int numofbar;
std_bar(char index1)
{
    int x, y, i;

```

```

switch(index1)
{
    case 'B':
        bar_func();
        break;

    case 'F':
        if(s_fact[0] != 0.0){
            make_graphw(49);
            bar_graphSF();
            escape_key();
            for(i=0;i<=numofbar;i++)
                rwind();
            return;
        }
    else{
        messa_sfact();
    }
    break;

    case 'C':
        curent_func();
        break;

    case 'S':
        standard_func();
        break;
}
return;
}

void bar_func()
{
    char *cases;
    int def,i;

    cases="try";
    def=1;
    while(cases != NULL){
        switch(def)
        {
            case 0:
                cases=NULL;
                break;
            case 1:
                make_graphw(77);
                bar_graph1();
                def = keyinkey(def);
                for(i=0;i<=numofbar;i++)
                    rwind();
                break;
            case 2:
                make_graphw(77);
                bar_graph2();
                def = keyinkey(def);
                for(i=0;i<=numofbar;i++)
                    rwind();
                break;
            case 3:
                make_graphw(77);
                bar_graph3();
                def = keyinkey(def);
                for(i=0;i<=numofbar;i++)
                    rwind();
        }
    }
}

```

```

        break;
    case 4:
        cases=NULL;
        break;
    }
    if(cases==NULL) break;
}

void bar_graph1()
{
    char *lable1, *lable2;
    int col,loc,hi,lo;
    lable1=
        "Chip Load          Chip Size          pH
Pressure";
    lable2=
        "          Chip Quality          Total SO2
Temperature ";
    numofbar=0;
    pwindf(68,0,attr(15,2),"%s","1 of 3");
    pwindf(7,20,attr(15,0),"%s",lable1);
    pwindf(7,21,attr(15,0),"%s",lable2);

    loc = load_range(database[0]);
    col=choose_color(bar_pos(load_time_limits(loc,1)));
    pwindf(10,1,attr(col,0),"%2.0f",database[0]); // chip load

    loc = total_range(database[3]);
    col=choose_color(bar_pos(total_time_limits(loc,1)));
    pwindf(39,1,attr(col,0),"%1.2f",database[3]); // total SO2

    loc = ph_range(database[4]);
    col=choose_color(bar_pos(ph_time_limits(loc,1)));
    pwindf(49,1,attr(col,0),"%1.1f",database[4]); // pH

    loc = temp_range(database[5]);
    col=choose_color(bar_pos(temp_time_limits(loc,1)));
    pwindf(59,1,attr(col,0),"%3.0f",database[5]); // temperature

    loc = press_range(database[6]);
    col=choose_color(bar_pos(press_time_limits(loc,1)));
    pwindf(69,1,attr(col,0),"%3.0f",database[6]); // pressure

    /**** BARS for each INPUT VARIABLES ****/

    //database[0] ----- CHIP LOAD
    loc = load_range(database[0]);
    hi=bar_pos(load_time_limits(loc,0));
    lo=bar_pos(load_time_limits(loc,1));
    make_bars(11,14,lo,hi);

    //database[1] ----- CHIP QUALITY
    loc = quality_range(database[1]);
    hi=bar_pos(quality_time_limits(loc,0));
    lo=bar_pos(quality_time_limits(loc,1));
    make_bars(21,24,lo,hi);

    //database[2] ----- CHIP SIZE
    loc = size_range(database[2]);
    hi=bar_pos(size_time_limits(loc,0));
    lo=bar_pos(size_time_limits(loc,1));

```

```

        make_bars(31,34,lo,hi);

//database[3] ----- TOTAL SO2
    loc = total_range(database[3]);
    hi=bar_pos(total_time_limits(loc,0));
    lo=bar_pos(total_time_limits(loc,1));
    make_bars(41,44,lo,hi);

//database[4] ----- pH
    loc = ph_range(database[4]);
    hi=bar_pos(ph_time_limits(loc,0));
    lo=bar_pos(ph_time_limits(loc,1));
    make_bars(51,54,lo,hi);

//database[5] ----- TEMPERATURE
    loc = temp_range(database[5]);
    hi=bar_pos(temp_time_limits(loc,0));
    lo=bar_pos(temp_time_limits(loc,1));
    make_bars(61,64,lo,hi);

//database[6] ----- PRESSURE
    loc = press_range(database[6]);
    hi=bar_pos(press_time_limits(loc,0));
    lo=bar_pos(press_time_limits(loc,1));
    make_bars(71,74,lo,hi);

}

void bar_graph2()
{
    char *label1;
    int lo,hi;
    label1=
        "          Wood Chips          Cooking Acid          Temp-
Pressure          ";
    numofbar=0;
    pwindf(68,0,attr(15,2),"%s","2 of 3");
    pwindf(7,21,attr(15,0),"%s",label1);

    /* bars for couple variables */

    //Wood Chips
        load_quali_size_couple(database[0],database[1],database[2]);
        hi=bar_pos(couple_time_limit[0][0]);
        lo=bar_pos(couple_time_limit[0][1]);
        make_bars(19,26,lo,hi);

    //Cooking Acid
        ph_total_couple(database[4],database[3]);
        hi=bar_pos(couple_time_limit[1][0]);
        lo=bar_pos(couple_time_limit[1][1]);
        make_bars(39,46,lo,hi);

    //Temp-Press
        press_temp_couple(database[6],database[5]);
        hi=bar_pos(couple_time_limit[2][0]);
        lo=bar_pos(couple_time_limit[2][1]);
        make_bars(59,66,lo,hi);

}

```

```

void bar_graph3()
{
    char *label1;
    int lo,hi;
    label1=
        "
                                All Process Input Variables
";
    numofbar=0;
    pwindf(68,0,attr(15,2),"s","3 of 3");
    pwindf(7,21,attr(15,0),"s",label1);

/* bars for couple variables */

//All input variables
load_quali_size_couple(database[0],database[1],database[2]);
ph_total_couple(database[4],database[3]);
press_temp_couple(database[6],database[5]);
all_couple();
hi=bar_pos(couple_time_limit[3][0]);
lo=bar_pos(couple_time_limit[3][1]);
make_bars(37,48,lo,hi);

}

bar_graphSF()
{
    char *label1;
    double ct;
    label1=
        "
                                S-Factor Model
";
    numofbar=0;
    wtitle(RIGHT,BOTTOM,attr(14,4),"
                                press <ESC>
");
    pwindf(7,21,attr(15,0),"s",label1);

//S-Factor Model
ct=reaction_rate();
if(ct >= 4.35000)
    {pwindf(35,1,attr(15,5),"1.2f",ct);
      pwindf(36,1,attr(15,5),":");
      make_bars(22,33,2,1);}
if(ct >= 4.30000 && ct < 4.35000)
    {pwindf(35,2,attr(15,5),"1.2f",ct);
      pwindf(36,2,attr(15,5),":");
      make_bars(22,33,3,2);}
if(ct >= 4.25000 && ct < 4.30000)
    {pwindf(35,3,attr(15,5),"1.2f",ct);
      pwindf(36,3,attr(15,5),":");
      make_bars(22,33,4,3);}
if(ct >= 4.20000 && ct < 4.25000)
    {pwindf(35,4,attr(15,5),"1.2f",ct);
      pwindf(36,4,attr(15,5),":");
      make_bars(22,33,5,4);}
if(ct >= 4.15000 && ct < 4.20000)
    {pwindf(35,5,attr(15,5),"1.2f",ct);
      pwindf(36,5,attr(15,5),":");
      make_bars(22,33,6,5);}
if(ct >= 4.10000 && ct < 4.15000)
    {pwindf(35,6,attr(15,5),"1.2f",ct);
      pwindf(36,6,attr(15,5),":");
      make_bars(22,33,7,6);}
if(ct >= 4.05000 && ct < 4.10000)
    {pwindf(35,7,attr(15,5),"1.2f",ct);
      pwindf(36,7,attr(15,5),":");

```

```

    makeBars(22,33,8,7);}
if(ct >= 4.00000 && ct < 4.05000)
    {pwindf(35,8,attr(15,5),"%1.2f",ct);
      pwindf(36,8,attr(15,5),":");
      makeBars(22,33,8,9);}
if(ct > 3.55000 && ct < 4.00000)
    {pwindf(35,9,attr(15,5),"%1.2f",ct);
      pwindf(36,9,attr(15,5),":");
      makeBars(22,33,9,10);}
if(ct == 3.55)
    {pwindf(35,9,attr(15,5),"%1.2f",ct);
      pwindf(36,9,attr(15,5),":");
      makeBars(22,33,10,10);}
if(ct >= 3.50000 && ct < 3.55000)
    {pwindf(35,10,attr(15,5),"%1.2f",ct);
      pwindf(36,10,attr(15,5),":");
      makeBars(22,33,11,10);}
if(ct >= 3.45000 && ct < 3.50000)
    {pwindf(35,11,attr(15,5),"%1.2f",ct);
      pwindf(36,11,attr(15,5),":");
      makeBars(22,33,12,11);}
if(ct >= 3.40000 && ct < 3.45000)
    {pwindf(35,12,attr(15,5),"%1.2f",ct);
      pwindf(36,12,attr(15,5),":");
      makeBars(22,33,13,12);}
if(ct >= 3.35000 && ct < 3.40000)
    {pwindf(35,13,attr(15,5),"%1.2f",ct);
      pwindf(36,13,attr(15,5),":");
      makeBars(22,33,14,13);}
if(ct >= 3.30000 && ct < 3.35000)
    {pwindf(35,14,attr(15,5),"%1.2f",ct);
      pwindf(36,14,attr(15,5),":");
      makeBars(22,33,15,14);}
if(ct >= 3.25000 && ct < 3.30000)
    {pwindf(35,15,attr(15,5),"%1.2f",ct);
      pwindf(36,15,attr(15,5),":");
      makeBars(22,33,16,15);}
if(ct >= 3.20000 && ct < 3.25000)
    {pwindf(35,16,attr(15,5),"%1.2f",ct);
      pwindf(36,16,attr(15,5),":");
      makeBars(22,33,17,16);}
if(ct >= 3.15000 && ct < 3.20000)
    {pwindf(35,17,attr(15,5),"%1.2f",ct);
      pwindf(36,17,attr(15,5),":");
      makeBars(22,33,18,17);}
if(ct >= 3.10000 && ct < 3.14000)
    {pwindf(35,18,attr(15,5),"%1.2f",ct);
      pwindf(36,18,attr(15,5),":");
      makeBars(22,33,19,18);}
if(ct >= 3.05000 && ct < 3.10000)
    {pwindf(35,19,attr(15,5),"%1.2f",ct);
      pwindf(36,19,attr(15,5),":");
      makeBars(22,33,20,19);}
if(ct < 3.05000)
    {pwindf(35,20,attr(15,5),"%1.2f",ct);
      pwindf(36,20,attr(15,5),":");
      makeBars(22,33,21,20);}
return;
}

```

```

void make_graphw(int x)
{
    char *timel, *timtit;

```

```

int i;
timtit="COOKING TIME (hrs)";
timel=
    "4:35-\n"
    "4:30-\n"
    "4:25-\n"
    "4:20-\n"
    "4:15-\n"
    "4:10-\n"
    "4:05-\n"
    "4:00-\n"
    "3:55-\n"
    "3:50-\n"
    "3:45-\n"
    "3:40-\n"
    "3:35-\n"
    "3:30-\n"
    "3:25-\n"
    "3:20-\n"
    "3:15-\n"
    "3:10-\n"
    "3:05-";

mwind(1,0,x,23,attr(0,0),attr(0,4),3);
wtitle(CENTER, TOP, attr(14,4), " BAR GRAPH FOR KAPPA NUMBER
CONTROL ");
wtitle(RIGHT, BOTTOM, attr(14,4), " press <ESC> or <Pg Up> or <Pg
Dn> ");
pwindf(1,1,attr(15,0), "%s", timel);
pwindf(0,0,attr(15,5), "%s", timtit);
mline(8,2,18,attr(15,0),vert,0);
mline(9,10,x-10,attr(5,1),hornz,5);
mline(9,11,x-10,attr(5,1),hornz,5);
mline(9,12,x-10,attr(5,1),hornz,5);

for(i=2;i<=9;i++)
    mline(9,i,x-10,attr(8,0),hornz,0);
for(i=13;i<=19;i++)
    mline(9,i,x-10,attr(8,0),hornz,0);
    mline(9,20,x-10,attr(15,0),hornz,0);
}

void makeBars(int x1, int x2, int yt, int ytt)
{
    int col;
    if (yt!=22 && ytt!=22){
        if ( yt==11 ){
            col=2;}
        else{
            col=4;}
    mwind(x1,yt,x2,20,attr(col,col),attr(col,col),5);
    numofbar++;
    if(ytt!=0 && ytt!=11 ){
        mwind(x1,ytt,x2,yt,attr(14,14),attr(14,14),5);
        numofbar++;}
    }
}

int bar_pos(char *st)
{
    int level;
    if(!strcmp(st,"+55"))    level=0;    // 4:40
    if(!strcmp(st,"+50"))    level=1;    // 4:40
    if(!strcmp(st,"+45"))    level=1;    // 4:40

```



```

        if(!strcmp(st,"+40"))    level=2;    // 4:35
        if(!strcmp(st,"+35"))    level=3;    // 4:30
        if(!strcmp(st,"+30"))    level=4;    // 4:25
        if(!strcmp(st,"+25"))    level=5;    // 4:20
        if(!strcmp(st,"+20"))    level=6;    // 4:15
        if(!strcmp(st,"+15"))    level=7;    // 4:10
        if(!strcmp(st,"+10"))    level=8;    // 4:05
        if(!strcmp(st,"+5"))     level=9;    // 4:00
        if(!strcmp(st,"+0"))     level=10;   // 3:55
        if(!strcmp(st,"GOOD"))   level=11;   // 3:50
        if(!strcmp(st,"-0"))     level=12;   // 3:45
        if(!strcmp(st,"-5"))     level=13;   // 3:40
        if(!strcmp(st,"-10"))    level=14;   // 3:35
        if(!strcmp(st,"-15"))    level=15;   // 3:30
        if(!strcmp(st,"-20"))    level=16;   // 3:25
        if(!strcmp(st,"-25"))    level=17;   // 3:20
        if(!strcmp(st,"-30"))    level=18;   // 3:15
        if(!strcmp(st,"-35"))    level=19;   // 3:10
        if(!strcmp(st,"-40"))    level=20;   // 3:05
        if(!strcmp(st,"-45"))    level=21;   // 3:00
        if(!strcmp(st,"-50"))    level=21;   // 2:55
        if(!strcmp(st,"-55"))    level=22;   // 2:50
    return level;
}

```

```

int choose_color(int pos)
{
    int col;

    if (pos==11)
        col=10;
    else
        col=12;
    return col;
}

```

```

void curent_func()
{
    int kein,i;

    char *desc;

    desc=
        " CURRENT INPUT VARIABLES CONDITIONS ";

    make_wind1(desc,database,0,4,14,5,7,15);
    escape_key();
    rwind();
}

```

```

void standard_func()
{
    int kein,i;
    char *desc;
    float normal[10];
    FILE *fp;
    desc=
        " STANDARD/NORMAL PULPING VARIABLES CONDITIONS ";

    if((fp=fopen("stdcond.knb","rb"))==NULL){
        printf("cannot open file\n");
    }
}

```

```

        exit(1);
    }

    fread(normal, sizeof(normal), 1, fp);

    make_wind1(desc,normal,7,4,14,5,0,1);
    escape_key();
    rwind();
}

void make_wind1(char *title, float data[10],
                int a1, int a2, int a3, int a4, int a5, int a6)
{
    char *size;

    mwind(2,1,65,23,attr(a1,a1),attr(a1,a2),3);
    wtitle(CENTER, TOP, attr(14,4), title);
    wtitle(CENTER, BOTTOM, attr(14,4), " PRESS <ESC> TO CONTINUE ");

    pwindf(7,1,attr(a3,a4)," W O O D   C H I P ");
    pwindf(7,3,attr(a5,a1),"CHIP LOAD                                Percent
gauge");
    pwindf(7,4,attr(a5,a1),"CHIP QUALITY");
    pwindf(7,5,attr(a5,a1),"CHIP SIZE");

    pwindf(7,8,attr(a3,a4)," C O O K I N G   A C I D ");
    pwindf(7,10,attr(a5,a1),"TOTAL SO2
Percent");
    pwindf(7,11,attr(a5,a1),"COMBINE SO2
Percent");
    pwindf(7,12,attr(a5,a1),"FREE SO2
Percent");
    pwindf(7,13,attr(a5,a1),"pH");

    pwindf(7,16,attr(a3,a4)," O T H E R   I N P U T S ");
    pwindf(7,18,attr(a5,a1),"OPERATING TEMPERATURE                degree
C");
    pwindf(7,19,attr(a5,a1),"OPERATING PRESSURE                    kPa");

    pwindf(32,3,attr(a6,a1),"%2.1f",data[0]);
    pwindf(32,4,attr(a6,a1),"%s",quality_selection(data[1]));
    pwindf(32,5,attr(a6,a1),"%s",size_selection(data[2]));
    pwindf(32,10,attr(a6,a1),"%1.2f",data[3]);
    pwindf(32,11,attr(a6,a1),"%1.2f",data[7]);
    pwindf(32,12,attr(a6,a1),"%1.2f",data[8]);
    pwindf(32,13,attr(a6,a1),"%1.1f",data[4]);
    pwindf(32,18,attr(a6,a1),"%3.0f",data[5]);
    pwindf(32,19,attr(a6,a1),"%3.0f",data[6]);
}

```

```

// *****
//
//          MODULE: "READDATA.C"
//
//
//
// *****

# include <stdio.h>
# include <stdlib.h>
extern float database[10];

void rw_stdcond(char *filename, int flag)
{
    FILE *fp;

    if(flag==0){
        if((fp=fopen(filename,"rb"))==NULL){
            printf("cannot open file\n");
            exit(1);
        }

        fread(database, sizeof(database), 1, fp);
    }

    if(flag==1){
        if((fp=fopen(filename,"wb"))==NULL){
            printf("cannot open file\n");
            exit(1);
        }

        fwrite(database, sizeof(database), 1, fp);
    }

    fclose(fp);
}

void set_stdcond()
{
    FILE *fp;
    float sample[10];
    if((fp=fopen("stdcond.knb","wb"))==NULL){
        printf("cannot open file\n");
        exit(1);
    }
    sample[0] = 38;          /* CHIP LOAD          */
    sample[1] = 1.0;         /* CHIP QUALITY       */
    sample[2] = 1.0;         /* CHIP SIZE          */
    sample[3] = 5.75;        /* TOTAL SO2          */
    sample[4] = 3.50;        /* pH                 */
    sample[5] = 163;         /* TEMPERATURE        */
    sample[6] = 680;         /* PRESSURE           */
    sample[7] = 2.80;        /* COMBINE SO2        */
    sample[8] = 2.95;        /* FREE SO2           */
    sample[9] = 0.00;        /* TOTAL SO2 (10)    */

    fwrite(sample, sizeof(sample), 1, fp);
    fclose(fp);
}

```

```

// *****
//
//          MODULE: "READTXT.C"
//
//
// *****

# include <stdio.h>
# include <stdlib.h>

char *readstuff (char *filename)
{
    FILE *fp;
    int count;
    char *txt, ch;

    if ( (fp=fopen(filename,"r")) !=NULL )
    {
        for ( count = 0 ; !feof(fp) ; count++)
            ch = getc(fp);

        if ( (txt = ( char * )malloc(count+1)) ==NULL){
            printf("Memory Allocation Error
digester\n");
            sleep(3);
            exit(-1);
        }

        rewind (fp);
        while (!feof(fp)){
            *txt = getc(fp);
            txt++;
        }
        *txt = 0;
    }

    else{
        printf ("error/n");
        exit(-1);
    }
    fclose(fp);
    return ( (char *) (txt - count) );
}

```

```

// *****
//
//          MODULE: "RULEBASE.C"
//
//
// *****

#include <stdarg.h>
#include <window.h>
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <string.h>
int rule_base_setA(float ,float [][][2]);
int rule_base_setB(float ,float []);
extern float database[10];

int rule_base_setA(float xx,float rang1[][2])
{
int i;

for(i=0;i<=9;i++){
    if (xx >= rang1[i][0] && xx <= rang1[i][1])
        break;
}
return i;
}

int rule_base_setB(float xx,float par[])
{
int i;

for(i=0;i<=9;i++){
    if(xx == par[i])
        break;
}
return i;
}

```

```

// *****
//
//          MODULE: "SIDEFUNC.C"
//
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>
# include <process.h>

void enter_key();
void escape_key();
void sound_func();
void pause();
void key_space_bar();
void please_wait();
int keyinkey(int loc1);
float change_to_min(float);
float change_to_hrs(float);

void escape_key()
{
    int kein;

    for(;;){
        kein = keyin();
        if(kein==ESC) break;
        sound_func(450);
    }
}

void sound_func(int freq)
{
    sound(freq);
    delay(100);
    nosound();
}

void pause()
{
    int kein;

    for(;;){
        kein = keyin();
        if(kein==CR) break;
        sound_func(450);
    }
}

void key_space_bar()
{
    int kein;

    for(;;){
        kein = keyin();
        if(kein==32) break;
    }
}

```

```

        sound_func(450);
    }
}
void please_wait()
{
    mwind(15,7,65,11,attr(15,4),attr(15,4),3);
    pwindfc(1,attr(15,4),"%s","One Moment Please !!!");
    return;
}

int keyinkey(int loc1)
{
    int x1;

    for(;;){
        x1=keyin();
        if(x1==329){ loc1--;
                    break;}
        if(x1==337){ loc1++;
                    break;}
        if(x1==27) { loc1=0;
                    break;}
        sound_func(1000);
    }
    clrwind(black);
    return loc1;
}
void enter_key()
{
    int kein;

    for(;;){
        kein = keyin();
        if(kein==CR) break;
        sound_func(450);
    }
}

float change_to_min(float tm)
{
    int hr;
    hr = 0;

    while(tm>=1.0){
        tm=tm-1.0;
        hr=hr+1.0;
    }
    tm=(60*hr)+(tm*100);
    return tm;
}

float change_to_hrs(float tm)
{
    int hr;
    hr = 0;
    tm=tm/60.0;
    while(tm>=1.0){
        tm=tm-1.0;
        hr=hr+1.0;
    }
    tm=hr+((tm*60)/100.0);
    return tm;
}

```

```

// *****
//
//          MODULE: "SITUATIO.C"
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

float change_to_hrs(float);
float change_to_min(float);
float get_valu(int,int,int,char *,float,float);
float reaction_rate();
float get_rate_press();
float s_fact_model(float);
void press_situ_func();
void temp_situ_func();
void ph_situ_func();
void total_situ_func();
void load_situ_func();
void cooking_situ_model();
void messa_sfact();
void time_kappa();
void make_screen(float valu, char *unt, char *name,
                 char *status, char *timereq, char
*situation, char *g);

void make_screen_str(char *valu, char *name, char *status,
                    char *timereq, char *situation );

void make_screen_fc(float valu1, float valu2, char *unt, char *name,
                    char *status, char *timereq, char
*situation, char *g);

int rule_base_setA(float xx,float rangel[][]);
char *quality_selection(float);
char *size_selection(float);

extern float s_fact[4];

extern float database[10];
extern char shadow;

int   ph_range(float);
char *ph_status(int);
char *ph_situation(int);
char *ph_time_required(int);

int   temp_range(float);
char *temp_status(int);
char *temp_situation(int);
char *temp_time_required(int);

int   press_range(float);
char *press_status(int);
char *press_situation(int);

```



```

char *press_time_required(int);

int  total_range(float);
char *total_status(int);
char *total_situation(int);
char *total_time_required(int);

int  load_range(float);
char *load_status(int);
char *load_situation(int);
char *load_time_required(int);

int  quality_range(float);
char *quality_status(int);
char *quality_situation(int);
char *quality_time_required(int);

int  size_range(float);
char *size_status(int);
char *size_situation(int);
char *size_time_required(int);

int  freecom_range(float, float);
char *freecom_status(int);
char *freecom_situation(int);
char *freecom_time_required(int);

situations(char index1)
{
    switch(index1)
    {
        case 'L':
            load_situ_func();
            break;

        case 'Q':
            quality_situ_func();
            break;

        case 'Z':
            size_situ_func();
            break;

        case 'T':
            total_situ_func();
            break;

        case 'F':
            free_situ_func();
            break;

        case 'H':
            ph_situ_func();
            break;

        case 'M':
            temp_situ_func();
            break;

        case 'P':
            press_situ_func();
            break;

        case 'C':

```

```

        cooking_situ_model();
        break;

    case 'K':
        time_kappa();
        break;

    }
return;
}

/* ----- C H I P L O A D ----- */
void load_situ_func()
{
    int loct;
    char *unit, *titl;
    unit=" % GAUGE";
    titl=" C H I P L O A D ";
    loct = load_range(database[0]);
    make_screen(database[0],unit,titl,load_status(loct),
load_time_required(loct),load_situation(loct),"%2.0f");
}

/* ----- C H I P Q U A L I T Y ----- */
quality_situ_func()
{
    int loct;
    char *titl,*select;
    titl=" C H I P Q U A L I T Y ";
    select = quality_selection(database[1]);
    loct = quality_range(database[1]);
    make_screen_str(select,titl,quality_status(loct),
                    quality_time_required(loct), quality_situation(loct));
return;
}

/* ----- C H I P S I Z E ----- */
size_situ_func()
{
    int loct;
    char *titl,*select;
    titl=" C H I P S I Z E ";
    select = size_selection(database[2]);
    loct = size_range(database[2]);
    make_screen_str(select,titl,size_status(loct),
                    size_time_required(loct),size_situation(loct));
return;
}

/* ----- S O 2 T O T A L ----- */
void total_situ_func()
{
    int loct;
    char *unit, *titl;
    unit="Percent";

```

```

    titl=" S O 2   T O T A L ";
    loct = total_range(database[3]);
    make_screen(database[3],unit,titl,total_status(loct),

total_time_required(loct),total_situation(loct),"%1.2f");
}

/* -----   F R E E & C O M B I N E   S O 2   ----- */

free_situ_func()
{
    int loct;
    char *unit, *titl;
    unit="Percent";
    titl=" FREE & COMBINE   SO2 ";
    loct = freecom_range(database[8],database[7]);
    make_screen_fc(database[8],database[7],unit,titl,freecom_status(loct),

        freecom_time_required(loct),freecom_situation(loct),"%1.2f");
    return;
}

/* -----   P H   ----- */

void ph_situ_func()
{
    int loct;
    char *unit, *titl;
    unit="";
    titl=" p H   o f   A C I D ";
    loct = ph_range(database[4]);
    make_screen(database[4],unit,titl,ph_status(loct),

ph_time_required(loct),ph_situation(loct),"%1.2f");
}

/* -----   T E M P E R A T U R E   ----- */

void temp_situ_func()
{
    int loct;
    char *unit, *titl;
    unit=" degree C";
    titl=" T E M P E R A T U R E ";
    loct = temp_range(database[5]);
    make_screen(database[5],unit,titl,temp_status(loct),

temp_time_required(loct),temp_situation(loct),"%1.2f");
}

/* -----   P R E S S U R E   ----- */

void press_situ_func()
{
    int loct;
    char *unit, *titl;
    unit=" kPa";
    titl=" P R E S S U R E ";
    loct = press_range(database[6]);
    make_screen(database[6],unit,titl,press_status(loct),

```

```

press_time_required(loct),press_situation(loct),"%1.2f");
}

/* ----- S - F A C T O R ----- */

void cooking_situ_model()
{
if(s_fact[0]==0.0)
    messa_sfact();

else{

    { mwind(10,1,40,5,attr(0,7),attr(15,7),3);
      pwindfc(0,attr(14,1),"%s"," S-FACTOR ");
      pwindfc(2,attr(0,7),"%4.2f",s_fact[0]);
      mwind(10,7,40,15,attr(14,10),attr(0,10),5);
      wtitle(CENTER,TOP,attr(0,10),"Estimated Cooking Time");
      pwindfc(0,attr(0,10),"%s","Base on S-FACTOR ");
      pwindf(6,2,attr(0,7),"%s","");
      pwindf(6,3,attr(0,7),"%s","");
      pwindf(6,4,attr(0,7),"%s","");
      pwindf(10,3,attr(0,7),"%1.2f",reaction_rate());
      pwindf(11,3,attr(0,7),"%s",":");
      pwindf(15,3,attr(0,7),"%s","Hrs");
    }

    wtitle(CENTER,BOTTOM,attr(14,10),"press <ESC> to quit");
    escape_key();
    rwind();
    rwind();
}
}

/* --- T I M E R E M A I N I N G - K A P P A N U M B E R ---- */

void time_kappa()
{ char *mess;
  float tim,hr;
  double s,k,kappa,tmin;
  if(s_fact[0]==0.0)
      messa_sfact();

else{
    hr=reaction_rate();
    mwind(10,1,40,5,attr(0,7),attr(15,7),3);
    pwindfc(0,attr(14,1),"%s"," S-FACTOR ");
    pwindfc(2,attr(0,7),"%4.2f",s_fact[0]);

    mwind(3,18,68,22,attr(0,0),attr(8,0),2);
    mess=
    "Enter TIME into cooking cycle .....[hr.min] -->";
    pwindf(1,1,attr(15,0),"%s",mess);
    tim = get_valu(51,1,3,"%1.2f",change_to_hrs(s_fact[2]),
                  (change_to_hrs(s_fact[3]+15.0)));
    if(tim==999.0){
        rwind();
        return;
    }
    pwindf(53,1,attr(0,7),":");
    pwindf(57,1,attr(0,7),"hrs");
    wtitle(CENTER,BOTTOM,attr(14,0),"Press << ENTER >>");
    enter_key();
    wtitle(CENTER,BOTTOM,attr(0,0),"");
}
}

```

```

mwind(10,6,40,17,attr(14,10),attr(0,10),5);
wtitle(CENTER,TOP,attr(0,10),"Estimated Remaining");
pwindfc(0,attr(0,10),"%s","Cooking Time");
pwindf(6,1,attr(0,7),"%s","");
pwindf(6,2,attr(0,7),"%s","");
pwindf(6,3,attr(0,7),"%s","");
hr=s_fact[3]-change_to_min(tim);

if(hr <= 0.0) {hr = 0.00;}
else          {hr = change_to_hrs(hr);}

pwindf(10,2,attr(0,7),"%.2f",hr);
pwindf(11,2,attr(0,7),"%s",".");
pwindf(15,2,attr(0,7),"%s","hrs");

pwindfc(5,attr(0,10),"%s","Estimated Current ");
pwindfc(6,attr(0,10),"%s","Kappa Number ");
pwindf(6,7,attr(0,7),"%s","");
pwindf(6,8,attr(0,7),"%s","");
pwindf(6,9,attr(0,7),"%s","");

k = get_rate_press();
tmin = change_to_min(tim) - s_fact[2];
s = k*(tmin/60.0) + s_fact[1];
kappa = s_fact_model(s);
pwindf(13,8,attr(0,7),"%.2f",kappa);

wtitle(CENTER,BOTTOM,attr(14,10),"press <ESC> to quit");
escape_key();
rwind();
rwind();
rwind();
rwind();
return;
}
}

void make_screen(float valu, char *unt, char *name,
                 char *status, char *timereq, char
                 *situation, char *g)
{
    mwind(10,1,40,5,attr(0,7),attr(15,7),3);
    pwindfc(0,attr(14,1),"%s",name);
    pwindf(8,2,attr(0,7),g,valu);
    pwindf(13,2,attr(0,7),"%s",unt);
    mwind(10,7,40,15,attr(14,10),attr(0,10),3);
    wtitle(CENTER,TOP,attr(15,10)," COOKING TIME ");
    pwindfc(0,attr(0,10),"%s",status);
    pwindfc(3,attr(0,10),"%s",timereq);
    wtitle(CENTER,BOTTOM,attr(4,10)," press <SPACE BAR> ");
    key_space_bar();
    movwind(49,14);
    wtitle(CENTER,BOTTOM,attr(15,10),"");
    mwind(1,5,50,23,attr(0,6),attr(7,6),3);
    wtitle(CENTER,TOP,attr(15,6)," SITUATION ");
    pwindf(1,0,attr(0,6),"%s",situation);
    wtitle(CENTER,BOTTOM,attr(14,6)," press <ESC> ");
    escape_key();
    rwind();
    rwind();
    rwind();
    rwind();
}

```

```

void make_screen_str(char *valu, char *name, char *status,
                    char *timereq, char *situation )
{
    mwind(10,1,40,5,attr(0,7),attr(15,7),3);
    pwindfc(0,attr(14,1),"%s",name);
    pwindfc(2,attr(0,7),"%s",valu);
    mwind(10,7,40,15,attr(14,10),attr(0,10),3);
    wtitle(CENTER,TOP,attr(15,7), " COOKING TIME ");
    pwindfc(0,attr(0,10),"%s",status);
    pwindfc(3,attr(0,10),"%s",timereq);
    wtitle(CENTER,BOTTOM,attr(4,7), " press <SPACE BAR> ");
    key_space_bar();
    movwind(49,14);
    wtitle(CENTER,BOTTOM,attr(15,7), " ");
    mwind(1,5,50,23,attr(0,6),attr(7,6),3);
    wtitle(CENTER,TOP,attr(15,6), " SITUATION ");
    pwindf(1,0,attr(0,6),"%s",situation);
    wtitle(CENTER,BOTTOM,attr(14,6), " press <ESC> ");
    escape_key();
    rwind();
    rwind();
    rwind();
}

void make_screen_fc(float valu1, float valu2, char *unt, char *name,
                    char *status, char *timereq, char
                    *situation, char *g)
{
    mwind(10,1,40,5,attr(0,7),attr(15,7),3);
    pwindfc(0,attr(14,1),"%s",name);
    pwindf(6,2,attr(0,7),g,valu1);
    pwindf(13,2,attr(0,7),g,valu2);
    pwindf(19,2,attr(0,7),"%s",unt);
    mwind(10,7,40,15,attr(14,10),attr(0,10),3);
    wtitle(CENTER,TOP,attr(15,7), " COOKING TIME ");
    pwindfc(0,attr(0,10),"%s",status);
    pwindfc(3,attr(0,10),"%s",timereq);
    wtitle(CENTER,BOTTOM,attr(4,7), " press <SPACE BAR> ");
    key_space_bar();
    movwind(49,14);
    wtitle(CENTER,BOTTOM,attr(15,7), " ");
    mwind(1,5,50,23,attr(0,6),attr(7,6),3);
    wtitle(CENTER,TOP,attr(15,6), " SITUATION ");
    pwindf(1,0,attr(0,6),"%s",situation);
    wtitle(CENTER,BOTTOM,attr(14,6), " press <ESC> ");
    escape_key();
    rwind();
    rwind();
    rwind();
}

void messa_sfact()
{
    shadow=TRUE;
    mwind(5,16,50,21,attr(0,6),attr(0,6),3);
    pwindfc(0,attr(14,6),"%s", "SYSTEM DOES NOT KNOW THE VALUE OF");
    pwindfc(1,attr(12,0),"%s", " S-FACTOR ");
    pwindfc(2,attr(15,6),"%s", "select INPUTS and S-FACTOR MODEL first");
    wtitle(CENTER,BOTTOM,attr(0,6), " press <ESC> ");
    escape_key();
    shadow = FALSE;
    rwind();
}

```

```

float reaction_rate()
{ float x,s,dt;

  s = get_rate_press();
  x = s_fact[0] - s_fact[1];
  dt = (x/s);
  dt = dt*60.0 + s_fact[2];
  s_fact[3]=dt;
  dt = change_to_hrs(dt);
  return dt;

}

float get_rate_press()
{ float k,x,s,y,p;

  y = 10166/(database[5] + 273.15);
  x = 27.255; // same as x = 10166/(100+273)
  k = exp(x-y);
  p = (database[6]+101.3)/101.3; // change into absolute atmospheric
pressure
  s = p*k; // pressure * reaction rate
  return s;
}

```

```

// *****
//
//          MODULE: "SPEC.C"
//
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

void window_spec(char *tit3);
extern char shadow;
char *readstuff(char *);
char *title,cases;
int def;

equip_spec(char index1)
{
    switch(index1)
    {
        case 'D':
            diges_spec();
            break;

        case 'A':
            acid_spec();
            break;

        case 'C':
            conden_spec();
            break;

        case 'W':
            water_spec();
            break;

        case 'R':
            draind_spec();
            break;

        case 'P':
            dump_spec();
            break;

        case 'F':
            flush_spec();
            break;
    }
    return;
}

diges_spec()
{
    char *digspecl, *digspec2;

    please_wait();
}

```



```

        title=
        " DIGESTERS SPECIFICATION ";

digspec1=readstuff("digest1.spe");
digspec2=readstuff("digest2.spe");
cases=' ';
def=1;
rwind();
window_spec(title);

while(cases != NULL){
switch(def)
{
    case 0:
        rwind();
        cases=NULL;
        break;
    case 1:
        pwindf(2,1,attr(14,0),"%s",digspec1);
        def = keyinkey(def);
        break;
    case 2:
        pwindf(2,1,attr(14,0),"%s",digspec2);
        def = keyinkey(def);
        break;
    case 3:
        rwind();
        cases=NULL;
        break;
}
    if(cases==NULL) break;
}
    free(digspec1);
    free(digspec2);
return;
}

```

```

acid_spec()
{
    char *acids1, *acids2, *acids3, *acids4;

    please_wait();
    title=
    " ACID SYSTEM SPECIFICATION ";

    acids1=readstuff("acids1.spe");
    acids2=readstuff("acids2.spe");
    acids3=readstuff("acids3.spe");
    acids4=readstuff("acids4.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
    switch(def)
    {
        case 0:
            rwind();
            cases=NULL;
            break;
        case 1:
            pwindf(2,1,attr(14,0),"%s",acids1);

```

```

        def = keyinkey(def);
        break;
    case 2:
        pwindf(2,1,attr(14,0),"%s",acids2);
        def = keyinkey(def);
        break;
    case 3:
        pwindf(2,1,attr(14,0),"%s",acids3);
        def = keyinkey(def);
        break;
    case 4:
        pwindf(2,1,attr(14,0),"%s",acids4);
        def = keyinkey(def);
        break;
    case 5:
        rwind();
        cases=NULL;
        break;
    }
    if(cases==NULL) break;
}
free(acids1);
free(acids2);
free(acids3);
free(acids4);
return;
}

conden_spec()
{
    char *condens1;

    please_wait();
    title=
    " CONDENSATE SYSTEM SPECIFICATION ";

    condens1=readstuff("condens1.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"%s",condens1);
                def = keyinkey(def);
                break;
            case 2:
                rwind();
                cases=NULL;
                break;
        }
        if(cases==NULL) break;
    }
    free(condens1);
return;
}

```

```

water_spec()
{
    char *water1, *water2;

    please_wait();
    title=
        " HOT & WARM WATER SYSTEM SPECIFICATION ";

    water1=readstuff("water1.spe");
    water2=readstuff("water2.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"%s",water1);
                def = keyinkey(def);
                break;
            case 2:
                pwindf(2,1,attr(14,0),"%s",water2);
                def = keyinkey(def);
                break;
            case 3:
                rwind();
                cases=NULL;
                break;
        }
        if(cases==NULL) break;
    }
    free(water1);
    free(water2);
return;
}

```

```

draind_spec()
{
    char *drain1, *drain2, *drain3, *drain4;

    please_wait();
    title=
        " DRAIN DOWN SYSTEM SPECIFICATION ";

    drain1=readstuff("drain1.spe");
    drain2=readstuff("drain2.spe");
    drain3=readstuff("drain3.spe");
    drain4=readstuff("drain4.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();

```

```

        cases=NULL;
        break;
    case 1:
        pwindf(2,1,attr(14,0),"%s",drain1);
        def = keyinkey(def);
        break;
    case 2:
        pwindf(2,1,attr(14,0),"%s",drain2);
        def = keyinkey(def);
        break;
    case 3:
        pwindf(2,1,attr(14,0),"%s",drain3);
        def = keyinkey(def);
        break;
    case 4:
        pwindf(2,1,attr(14,0),"%s",drain4);
        def = keyinkey(def);
        break;
    case 5:
        rwind();
        cases=NULL;
        break;
    }
    if(cases==NULL) break;
}
free(drain1);
free(drain2);
free(drain3);
free(drain4);
return;
}

dump_spec()
{
    char *dump1, *dump2, *dump3, *dump4;

    please_wait();
    title=
    " DUMP SYSTEM SPECIFICATION ";

    dump1=readstuff("dump1.spe");
    dump2=readstuff("dump2.spe");
    dump3=readstuff("dump3.spe");
    dump4=readstuff("dump4.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"%s",dump1);
                def = keyinkey(def);
                break;
            case 2:
                pwindf(2,1,attr(14,0),"%s",dump2);
                def = keyinkey(def);
                break;

```

```

        case 3:
            pwindf(2,1,attr(14,0),"%s",dump3);
            def = keyinkey(def);
            break;
        case 4:
            pwindf(2,1,attr(14,0),"%s",dump4);
            def = keyinkey(def);
            break;
        case 5:
            rwind();
            cases=NULL;
            break;
    }
    if(cases==NULL) break;
}
free(dumpl);
free(dump2);
free(dump3);
free(dump4);
return;
}

flush_spec()
{
    char *flush1, *flush2;

    please_wait();
    title=
    " FLUSHING LIQUOR SYSTEM SPECIFICATION ";

    flush1=readstuff("flush1.spe");
    flush2=readstuff("flush2.spe");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"%s",flush1);
                def = keyinkey(def);
                break;
            case 2:
                pwindf(2,1,attr(14,0),"%s",flush2);
                def = keyinkey(def);
                break;
            case 3:
                rwind();
                cases=NULL;
                break;
        }
        if(cases==NULL) break;
    }
    free(flush1);
    free(flush2);
    return;
}

```

```

pulping_inf()
{
    char *pulp1, *pulp2;

    please_wait();
    title=
        " CHEMICAL (SULPHITE) PULPING ";

    pulp1=readstuff("pulping1.inf");
    pulp2=readstuff("pulping2.inf");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"s",pulp1);
                def = keyinkey(def);
                break;
            case 2:
                pwindf(2,1,attr(14,0),"s",pulp2);
                def = keyinkey(def);
                break;
            case 3:
                rwind();
                cases=NULL;
                break;
        }
        if(cases==NULL) break;
    }
    free(pulp1);
    free(pulp2);

    return;
}

void window_spec(char *tit3)
{
    mwind(2,1,75,21,attr(2,0),attr(7,6),3);
    wtitle(CENTER,TOP,attr(15,6),tit3);
    wtitle(RIGHT,BOTTOM,attr(15,6)," PRESS <ESC> or <Pg Up> or <Pg
Dn> ");
}

opera_steps()
{
    char *step1, *step2, *step3, *step4, *step5;
    char *step6, *step7, *step8, *step9;

    please_wait();
    title=
        " DIGESTER OPERATION PROCEDURES ";

    step1=readstuff("step1.txx");
    step2=readstuff("step2.txx");
    step3=readstuff("step3.txx");

```

```

step4=readstuff("step4.txx");
step5=readstuff("step5.txx");
step6=readstuff("step6.txx");
step7=readstuff("step7.txx");
step8=readstuff("step8.txx");
step9=readstuff("step9.txx");
cases=' ';
def=1;
rwind();
window_spec(title);
while(cases != NULL){
switch(def)
{
case 0:
rwind();
cases=NULL;
break;
case 1:
pwindf(2,1,attr(14,0),"s",step1);
def = keyinkey(def);
break;
case 2:
pwindf(2,1,attr(14,0),"s",step2);
def = keyinkey(def);
break;
case 3:
pwindf(2,1,attr(14,0),"s",step3);
def = keyinkey(def);
break;
case 4:
pwindf(2,1,attr(14,0),"s",step4);
def = keyinkey(def);
break;
case 5:
pwindf(2,1,attr(14,0),"s",step5);
def = keyinkey(def);
break;
case 6:
pwindf(2,1,attr(14,0),"s",step6);
def = keyinkey(def);
break;
case 7:
pwindf(2,1,attr(14,0),"s",step7);
def = keyinkey(def);
break;
case 8:
pwindf(2,1,attr(14,0),"s",step8);
def = keyinkey(def);
break;
case 9:
pwindf(2,1,attr(14,0),"s",step9);
def = keyinkey(def);
break;
case 10:
rwind();
cases=NULL;
break;
}
if(cases==NULL) break;
}
free(step1);
free(step2);
free(step3);
free(step4);

```

```

        free(step5);
        free(step6);
        free(step7);
        free(step8);
        free(step9);
    return;
}

introduc()
{ char *intro1, *intro2, *intro3, *intro4;

    please_wait();
    title=
    " INTRODUCTION ";

    intro1=readstuff("intro1.txx");
    intro2=readstuff("intro2.txx");
    intro3=readstuff("intro3.txx");
    intro4=readstuff("intro4.txx");
    cases=' ';
    def=1;
    rwind();
    window_spec(title);

    while(cases != NULL){
        switch(def)
        {
            case 0:
                rwind();
                cases=NULL;
                break;
            case 1:
                pwindf(2,1,attr(14,0),"%s",intro1);
                def = keyinkey(def);
                break;
            case 2:
                pwindf(2,1,attr(14,0),"%s",intro2);
                def = keyinkey(def);
                break;
            case 3:
                pwindf(2,1,attr(14,0),"%s",intro3);
                def = keyinkey(def);
                break;
            case 4:
                pwindf(2,1,attr(14,0),"%s",intro4);
                def = keyinkey(def);
                break;
            case 5:
                rwind();
                cases=NULL;
                break;
        }
        if(cases==NULL) break;
    }
    free(intro1);
    free(intro2);
    free(intro3);
    free(intro4);
    return;
}

```



```

// *****
//
//          MODULE: "STARSHUT.C"
//
//
// *****

# include <stdarg.h>
# include <window.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>
# include <math.h>

extern char shadow;
char *readstuff(char *);
start_up()
{ char *cases;
  int def;
  char *start1, *start2, *start3, *start4, *start5;
  char *start6, *start7, *start8;

  shadow=TRUE;
  please_wait();
  start1=readstuff("start1.knb");
  start2=readstuff("start2.knb");
  start3=readstuff("start3.knb");
  start4=readstuff("start4.knb");
  start5=readstuff("start5.knb");
  start6=readstuff("start6.knb");
  start7=readstuff("start7.knb");
  start8=readstuff("start8.knb");
  rwind();

  mwind(5,1,73,21,attr(2,0),attr(7,6),3);
  wtitle(CENTER,TOP,attr(15,6)," START-UP PROCEDURES ");
  wtitle(RIGHT,BOTTOM,attr(15,6)," PRESS <ESC> or <Pg Up> or <Pg
Dn> ");
  cases="try";
  def=1;
  while(cases != NULL){
    switch(def)
    {
      case 0:
        rwind();
        cases=NULL;
        break;
      case 1:
        pwindf(4,1,attr(14,0),"s",start1);
        def = keyinkey(def);
        break;
      case 2:
        pwindf(4,1,attr(14,0),"s",start2);
        def = keyinkey(def);
        break;
      case 3:
        pwindf(4,1,attr(14,0),"s",start3);
        def = keyinkey(def);
        break;
      case 4:
        pwindf(4,1,attr(14,0),"s",start4);

```

```

        def = keyinkey(def);
        break;
    case 5:
        pwindf(4,1,attr(14,0),"s",start5);
        def = keyinkey(def);
        break;
    case 6:
        pwindf(4,1,attr(14,0),"s",start6);
        def = keyinkey(def);
        break;
    case 7:
        pwindf(4,1,attr(14,0),"s",start7);
        def = keyinkey(def);
        break;
    case 8:
        pwindf(4,1,attr(14,0),"s",start8);
        def = keyinkey(def);
        break;
    case 9:
        rwind();
        cases=NULL;
        break;
    }
    if(cases==NULL) break;
}
free(start1);
free(start2);
free(start3);
free(start4);
free(start5);
free(start6);
free(start7);
free(start8);
shadow=FALSE;
return;
}

shut_down()
{ char *menu2, index2;
  int see;
  menu2 =
    "\n"
    "Planned shutdown \n"
    "Short shutdown \n"
    "Emergency shutdown\n"
    "PSE\n";

  index2=popupmenu(37,1,menu2,2);
  switch(index2){
    case 'P':
      rwind();
      planned();
      break;

    case 'S':
      rwind();
      shortdown();
      break;

    case 'E':
      rwind();
      emergency();
      break;
  }
}

```

```

    }
    if(index2==NULL)
        {rwind();
        return;}
return;
}

planned()
{
    int def2;
    char cas2, index3;
    char *shut1, *shut2, *shut3;
    char *plan;

    shut1 = readstuff("shut1.knb");
    shut2 = readstuff("shut2.knb");
    shut3 = readstuff("shut3.knb");
    plan=
    "The planned shutdown is a procedures whereby\n"
    "the whole system is shutdown, either with\n"
    "all tanks empty or full depending on the\n"
    "situation desired.\n";

    shadow=TRUE;
    mwind(5,3,70,10,attr(2,0),attr(0,2),3);
    wtitle(CENTER,TOP,attr(15,2)," PLANNED SHUTDOWN ");
    pwindfc(1,attr(14,0),"s",plan);
    wtitle(CENTER,BOTTOM,attr(15,2)," PRESS <SPACE BAR> TO CONTINUE
");
    key_space_bar();
    rwind();
    mwind(5,1,73,22,attr(2,0),attr(7,6),3);
    wtitle(CENTER,TOP,attr(15,6)," PLANNED SHUTDOWN PROCEDURES ");
    wtitle(RIGHT,BOTTOM,attr(15,6)," PRESS <ESC> or <Pg Up> or <Pg
Dn> ");
    cas2="try";
    def2=1;
    while(cas2 != NULL){
        switch(def2){

            case 0:
                cas2=NULL;
                break;

            case 1:
                pwindf(4,1,attr(14,0),"s",shut1);
                def2 = keyinkey(def2);
                break;

            case 2:
                pwindf(4,1,attr(14,0),"s",shut2);
                def2 = keyinkey(def2);
                break;

            case 3:
                pwindf(4,1,attr(14,0),"s",shut3);
                def2 = keyinkey(def2);
                break;

            case 4:
                cas2=NULL;
                break;

        }
        if(cas2==NULL){
            rwind();
            break;
        }
    }
}

```

```

    }
    free(shut1);
    free(shut2);
    free(shut3);
    shadow=FALSE;
return;
}

shutdown()
{
    char *shor;

    shor=
    "The short shutdown is mainly necessary in case\n"
    "of mechanical trouble and usually affects only\n"
    "a portion of the system. The shutdown procedure\n"
    "in this case is the same as in the PLANNED SHUTDOWN.\n"
    "\n"
    "It is usually sufficient to stop only the parts of\n"
    "the system which are ahead of the equipment\n"
    "requiring repair.\n";
    shadow=TRUE;
    mwind(5,3,70,14,attr(2,0),attr(0,2),3);
    wtitle(CENTER, TOP, attr(15,2), " SHORT SHUTDOWN ");
    pwindfc(1, attr(14,0), "%s", shor);
    wtitle(CENTER, BOTTOM, attr(15,2), " PRESS <SPACE BAR> TO CONTINUE
");
    key_space_bar();
    rwind();
    shadow=FALSE;
return;
}

emergency()
{
    char *emerg;

    emerg=
    "For the emergency shutdown all digesters are shut\n"
    "in and held, insofar as possible, at the point in the\n"
    "cooking cycle where they were at the time of the shutdown\n"
    "\n"
    "The acid plant, condensate system, drain down system,\n"
    "flushing system, dumping system and warm water system will\n"
    "then be shutdown in the same way as in a PLANNED SHUTDOWN\n";

    shadow=TRUE;
    mwind(5,3,70,14,attr(2,0),attr(0,2),3);
    wtitle(CENTER, TOP, attr(15,2), " EMERGENCY SHUTDOWN ");
    pwindfc(1, attr(14,0), "%s", emerg);
    wtitle(CENTER, BOTTOM, attr(15,2), " PRESS <SPACE BAR> TO CONTINUE
");
    key_space_bar();
    rwind();
    shadow=FALSE;
return;
}

```

```

// *****
//
//          MODULE: "PAGE1.C"
//
//
// *****

#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>

void box(int,int,int,int,int);
void set_screen1();
void set_window();
void restmode();

void main()
{
    clrscr();
    set_screen1();
    textattr (7);
    clrscr();
}

void set_screen1()
{
    int keim;
    int gdriver, gmode;
    register int i;
    gdriver = DETECT;
    initgraph(&gdriver,&gmode, "");

    setbkcolor(4);
    box(1,1,635,475,15);
    box(2,2,634,474,15);
    box(3,3,633,473,15);
    box(4,4,632,472,15);
    box(5,5,631,471,15);
    box(80,200,540,305,15);
    box(81,201,539,304,15);
    box(84,204,536,301,15);
    box(85,205,535,300,15);
    setfillstyle(9,6);
    floodfill(89,206,15);
    settextstyle(1,0,8);
    setcolor(10);

    outtextxy(90,20,"FRASER INC");
    settextstyle(1,0,3);
    setcolor(10);
    outtextxy(220,120,"SULPHITE MILL");
    settextstyle(2,0,7);
    setcolor(15);
    outtextxy(90,215,"    INTELLIGENT OPERATION SUPPORT ");
    outtextxy(90,240,"                SYSTEM ");
    outtextxy(90,265,"                FOR THE BATCH DIGESTERS");
    setcolor(14);
    outtextxy(90,330," Intelligence Engineering Laboratory");
    outtextxy(90,350," Department of Chemical Engineering");
    outtextxy(90,370," University of Alberta");
}

```

```

        outtextxy(90,390,"                Edmonton, Alberta ");
        setcolor(15);
        settextstyle(5,0,3);
        outtextxy(90,450,"                >>>>> PRESS ANY KEY TO CONTINUE <<<<<
");
        settextstyle(2,0,7);
        kein = getch();
        restorecrtmode();
        closegraph();
        return;
    }

void box(int startx,int starty,int endx,int endy,int color)
{
    setcolor(color);
    line(startx,starty,startx,endy);
    line(startx,starty,endx,starty);
    line(endx,starty,endx,endy);
    line(endx,endy,startx,endy);
}

```

```

// *****
//
//          MODULE: "FLOWCHAR.C"
//
//
//
// *****

#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<graphics.h>
#include<dos.h>
void lables_tempres();
void lables_chips();
void lables_acid();
void pump_draw();
void set_screeng(void);
void set_digester(void);
void flowrates(void);
void arrows(int,int,int,char,int);
void write_lables(void);
void r_stdcond(char *);
char *quality_selection(float);
char *size_selection(float);
float datab[10];
// flowchart()
main()
{char kein;
int driver, mode;
register int i;
driver = DETECT;
r_stdcond("currcond.dat");
clrscr();
initgraph(&driver,&mode," ");
set_screeng();
set_digester();
pump_draw();
flowrates();
write_lables();
lables_tempres();
lables_acid();
lables_chips();
for(;;){
kein=getche();
if(kein=='Q') break;
if(kein=='q') break;
}
restorecrtmode();
closegraph();
textattr(7);
clrscr();
return;
}
void set_screeng()
{
setbkcolor(0);
setcolor(15);
rectangle(0,0,639,479);
rectangle(1,1,638,478);
}

void set_digester()

```

```

(
  line(160,1,160,250);
  line(161,1,161,250);
  line(162,1,162,250);
  line(1,60,161,60);
  line(1,59,161,59);
  line(1,58,161,58);
  setfillstyle(1,7);
  floodfill(150,40,15);
  line(1,250,161,250);
  line(1,249,161,249);
  line(1,248,161,248);
  setfillstyle(1,1);
  floodfill(150,200,15);
  setcolor(14);
  rectangle(30,81,130,88);
  setfillstyle(1,14);
  floodfill(100,85,14);
  arrows(105,84,2,'E',4);
  arrows(65,84,2,'E',4);
  setcolor(14);
  rectangle(60,131,100,138);
  setfillstyle(1,14);
  floodfill(90,135,14);
  setcolor(13);
  rectangle(73,129,87,140);
  rectangle(72,128,88,141);
  line(73,129,87,140);
  line(72,128,88,141);
  line(73,140,87,129);
  line(72,141,88,128);

  settextstyle(3,0,2);
  setcolor(4);
  outtextxy(18,3,"  DIGESTER  ");
  outtextxy(10,28,"FLOW DIAGRAM");

  //  *** screen size in graphic mode ***
  //
  //          0-----639
  //          |
  //          |
  //          |
  //          |
  //          479
  //

  setcolor(14);
  setfillstyle(1,14);
  rectangle(279,95,441,102);
  rectangle(499,95,534,102);
  rectangle(408,388,453,395);
  floodfill(300,100,14);
  floodfill(500,100,14);
  floodfill(425,390,14);
  setcolor(15);
  line(517,130,517,350);  // digester
  line(518,130,518,350);
  line(519,130,519,350);
  line(520,130,520,350);
  line(420,130,420,350);
  line(421,130,421,350);

```



```

line(422,130,422,350);
line(423,130,423,350);
line(495,90,520,130);
line(494,90,519,130);
line(493,90,518,130); /* TOP OF DIGESTER */
line(492,90,517,130);
line(445,90,420,130);
line(446,90,421,130);
line(447,90,422,130);
line(448,90,423,130);
line(455,400,420,350);
line(456,400,421,350);
line(457,400,422,350);
line(458,400,423,350);
line(485,400,520,350); /* BOTTOM OF DIGESTER */
line(484,400,519,350);
line(483,400,518,350);
line(482,400,517,350);
rectangle(440,90,500,88);
rectangle(445,65,495,90);
rectangle(450,400,490,402);
rectangle(440,65,500,67);
setfillstyle(9,6);
floodfill(455,70,15);
setfillstyle(1,6);
floodfill(480,250,15);

// strainer
rectangle(423,280,517,305);
setfillstyle(8,15);
floodfill(500,285,15);

// piping
settextstyle(1,0,4);
setcolor(14);
rectangle(520,291,535,297);
rectangle(535,291,541,320);
rectangle(405,320,541,326);
rectangle(405,291,420,297);
rectangle(399,291,405,431);
rectangle(351,431,405,437);
rectangle(210,419,318,425);
rectangle(272,378,279,419);
rectangle(272,95,279,227);
rectangle(399,102,405,127);
rectangle(405,120,541,127);
rectangle(534,95,541,120);
rectangle(467,98,473,120);
rectangle(380,102,387,420);
rectangle(387,413,400,420);
rectangle(387,388,396,395);
rectangle(210,468,387,475);
rectangle(380,438,387,468);
setcolor(15);
setfillstyle(1,14);
floodfill(530,295,14);
floodfill(537,300,14);
floodfill(450,324,14);
floodfill(410,295,14);
floodfill(400,295,14);
floodfill(400,435,14);
floodfill(300,420,14);
floodfill(275,417,14);
floodfill(275,100,14);
floodfill(400,120,14);
floodfill(500,121,14);

```

```

floodfill(540,125,14);
floodfill(470,110,14);
floodfill(385,400,14);
floodfill(390,415,14);
floodfill(390,390,14);
floodfill(300,470,14);
floodfill(385,450,14);
floodfill(535,100,14);
setcolor(13);
rectangle(252,416,268,428);
rectangle(253,417,267,427);
line(252,416,268,428);
line(253,417,267,427);
line(252,428,268,416);
line(253,427,267,417);
rectangle(377,446,390,458);
rectangle(378,447,389,457);
line(377,446,390,458);
line(378,447,389,457);
line(377,458,390,446);
line(378,457,389,447);
setcolor(8);
circle(470,99,4);
setcolor(7);
circle(470,99,5);
circle(470,99,6);
setfillstyle(1,8);
floodfill(470,99,8);
setcolor(15);
line(457,400,457,445);
line(458,400,458,445);
line(477,465,638,465);
line(477,464,638,464);
arc(477,445,180,270,20);
arc(477,445,180,270,19);
line(482,400,482,430);
line(483,400,483,430);
line(492,440,638,440);
line(492,441,638,441);
arc(493,430,180,270,10);
arc(493,430,180,270,11);
setfillstyle(1,8);
floodfill(600,450,15);
setcolor(3);
rectangle(455,403,485,427);
setfillstyle(1,3);
floodfill(460,415,3);
setcolor(15);
line(454,405,486,405);
line(454,406,486,406);
line(454,425,486,425);
line(454,424,486,424);
line(453,403,483,423);
line(455,405,485,425);
line(454,404,486,426);
line(455,424,484,406);
line(455,425,485,405);
line(454,426,486,404);

/*      HEAT EXCHANGER      */
setcolor(15);
line(260,240,260,360);
line(259,240,259,360);
line(258,240,258,360);

```

```

    line(290,240,290,360);
    line(291,240,291,360);
    line(292,240,292,360);
    arc(275,240,0,180,15);
    arc(275,240,0,180,16);
    arc(275,240,0,180,17);
    arc(275,360,180,0,15);
    arc(275,360,180,0,16);
    arc(275,360,180,0,17);
    setfillstyle(1,4);
    floodfill(275,300,15);
    line(255,240,295,240);
    line(255,243,295,243);
    line(255,357,295,357);
    line(255,360,295,360);
    setfillstyle(9,4);
    floodfill(275,300,15);
    line(245,255,258,255);
    line(245,263,258,263);
    line(245,338,258,338);
    line(245,345,258,345);
    rectangle(242,252,245,266);
    rectangle(242,335,245,348);
    line(210,255,242,255);
    line(210,263,242,263);
    line(210,338,242,338);
    line(210,345,242,345);
    return;
}
void pump_draw()
{
    circle(355,435,15);
    circle(355,435,16);
    line(339,465,371,465);
    line(339,464,371,464);
    line(345,448,339,464);
    line(346,448,340,464);
    line(365,448,371,464);
    line(364,448,370,464);
    line(320,419,355,419);
    line(320,425,345,425);
    rectangle(318,416,320,427);
    setfillstyle(1,4);
    floodfill(319,420,15);
    setfillstyle(1,2);
    floodfill(355,435,15);
    setfillstyle(1,2);
    floodfill(355,460,15);
    setfillstyle(1,2);
    floodfill(330,422,15);
}
void flowrates()
{
    arrows(406,294,2,'W',4);
    arrows(532,294,2,'E',4);
    arrows(538,314,2,'S',4);
    arrows(402,314,2,'S',4);
    arrows(526,323,2,'W',4);
    arrows(486,323,2,'W',4);
    arrows(446,323,2,'W',4);
    arrows(406,323,2,'W',4);
    arrows(402,354,2,'S',4);
    arrows(402,394,2,'S',4);
}

```

```

arrows(402,434,2,'S',4);
arrows(300,422,2,'W',4);
arrows(275,392,2,'N',4);
arrows(275,190,2,'N',4);
arrows(275,150,2,'N',4);
arrows(275,110,2,'N',4);
arrows(305,98,2,'E',4);
arrows(345,98,2,'E',4);
arrows(385,98,2,'E',4);
arrows(425,98,2,'E',4);
arrows(515,98,2,'W',4);
arrows(510,124,2,'E',4);
arrows(470,124,2,'E',4);
arrows(430,124,2,'E',4);
arrows(402,115,2,'S',4);
arrows(470,108,2,'N',4);
arrows(537,108,2,'N',4);
arrows(384,140,2,'S',4);
arrows(384,180,2,'S',4);
arrows(384,220,2,'S',4);
arrows(384,260,2,'S',4);
arrows(384,300,2,'S',4);
arrows(384,340,2,'S',4);
arrows(384,380,2,'S',4);
arrows(395,392,2,'E',4);
arrows(430,392,2,'E',4);
arrows(395,416,2,'E',4);
}

void arrows(int x1,int y1,int s,char d,int col)
( int x2,x3,y2,y3,x4,y4;

switch(d){
case 'W':
x2 = x1 + 4*s;
y2 = y1 - s;
x3 = x2;
y3 = y1 + s;
x4 = x1 + 3;
y4 = y1;
break;
case 'E':
x2 = x1 - 4*s;
y2 = y1 - s;
x3 = x2;
y3 = y1 + s;
x4 = x1 - 3;
y4 = y1;
break;
case 'S':
x2 = x1 + s;
y2 = y1 - 4*s;
x3 = x1 - s;
y3 = y2;
x4 = x1;
y4 = y1 - 3;
break;
case 'N':
x2 = x1 + s;
y2 = y1 + 4*s;
x3 = x1 - s;
y3 = y2;
x4 = x1;

```

```

        y4 = y1 + 3;
        break;
    }
    setcolor(col);
    setfillstyle(1,col);
    line(x1,y1,x2,y2);
    line(x1,y1,x3,y3);
    line(x2,y2,x3,y3);
    floodfill(x4,y4,col);
}

void write_labels()
{
    setcolor(15);
    settextstyle(5,0,6);
    outtextxy(438,155,"DIGESTER");
    outtextxy(200,285,"LIQUOR");
    outtextxy(200,300,"HEATER");
    outtextxy(390,77,"PACKER");
    outtextxy(495,415,"DUMP VALVE");
    settextstyle(2,0,5);
    outtextxy(425,10," WOOD CHIPS ");
    outtextxy(421,25," INLET ");
    settextstyle(2,0,4);
    outtextxy(550,72,"PACKING STEAM");
    outtextxy(550,82," 50 psi");
    outtextxy(175,225,"STEAM IN");
    outtextxy(175,235,"150 psi");
    outtextxy(175,355,"CONDENSATE");
    outtextxy(177,365," OUT ");
    outtextxy(300,395,"RECIRCULATION");
    outtextxy(320,405,"PUMP");

    outtextxy(520,446,"PULP TO ROCK TRAP");
    outtextxy(185,110,"FLUSHING &");
    outtextxy(185,120,"QUENCHING");
    outtextxy(185,130,"LIQUOR");
    outtextxy(565,317,"FLUSHING &");
    outtextxy(565,327,"QUENCHING");
    outtextxy(565,337,"LIQUOR");
    outtextxy(50,92,"RECIRCULATED");
    outtextxy(65,102,"LIQUOR");
    outtextxy(65,150,"VALVE");
    outtextxy(426,305,"STRAINER PLATES");
    settextstyle(2,0,5);
    outtextxy(175,401,"SPENT LIQUOR");
    outtextxy(175,450,"RAW COOKING ACID");
    setcolor(14);
    outtextxy(8,200,"PRESS <<Q>> TO QUIT");

    setcolor(6);
    arrows(460,57,3,'S',6);
    arrows(470,57,3,'S',6);
    arrows(480,57,3,'S',6);
    arrows(499,78,3,'W',13);
    line(505,77,540,77);
    line(505,78,540,78);
    line(505,79,540,79);
    arrows(215,259,3,'E',13);
    line(175,258,213,258);
    line(175,259,213,259);
    line(175,260,213,260);
    arrows(175,341,3,'W',9);
    line(177,340,215,340);

```

```

line(177,341,215,341);
line(177,342,215,342);

arrows(235,472,3,'E',12);
line(161,471,233,471);
line(161,472,233,472);
line(161,473,233,473);
arrows(175,422,3,'W',7);
line(178,421,235,421);
line(178,422,235,422);
line(178,423,235,423);
arrows(272,102,4,'E',2);
line(220,102,270,102);
line(220,101,270,101);
line(220,103,270,103);
arrows(542,310,4,'W',2);
line(545,310,600,310);
line(545,309,600,309);
line(545,311,600,311);

}

void labes_acid()
{static char *total1, *co, *freel, *ph1;
 int dec,sign;

 setcolor(12);
 setviewport(3,414,162,476,1);
 rectangle(0,0,158,61);
 rectangle(1,1,159,62);
 setfillstyle(1,8);
 floodfill(8,50,12);
 settextstyle(0,0,1);
 setcolor(15);
 outtextxy(5,5,"TOTAL % SO2:");
 outtextxy(5,20,"FREE % SO2:");
 outtextxy(5,35,"COMBINE % SO2:");
 outtextxy(5,50,"pH:");
 settextstyle(2,0,5);
 setcolor(14);
 gcvt(datab[3],4,total1);
 outtextxy(125,1,total1);
 gcvt(datab[8],4,freel);
 outtextxy(125,16,freel);
 gcvt(datab[7],4,co);
 outtextxy(125,31,co);
 gcvt(datab[4],2,ph1);
 outtextxy(125,45,ph1);
}

void labes_chips()
{static char *load1;
 int dec,sign;

 setcolor(6);
 setviewport(165,5,430,66,1);
 rectangle(0,0,265,61);
 rectangle(1,1,264,60);
 setfillstyle(1,8);
 floodfill(200,50,6);
 settextstyle(0,0,1);
 setcolor(15);
 outtextxy(5,10,"LOAD (% gauge):");
 outtextxy(5,28,"QUALITY:");

```

```

    outtextxy(5,45,"SIZE:");
    settextstyle(2,0,5);
    setcolor(14);
    load1 = fcvt(datab[0], 0, &dec, &sign);
    outtextxy(150,5,load1);
    settextstyle(2,0,4);
    outtextxy(75,25,quality_selection(datab[1]));
    outtextxy(75,40,size_selection(datab[2]));
}

void lables_tempres()
{
    static char *tem,*pres;
    static char *t,*p;
    int dec,sign;
    setcolor(2);
    setviewport(430,195,510,260,1);
    rectangle(0,0,80,65);
    rectangle(1,1,79,64);
    setfillstyle(1,8);
    floodfill(50,50,2);
    setcolor(14);
    settextstyle(2,0,5);
    t="C";
    p="kPa";
    outtextxy(40,15,t);
    outtextxy(40,35,p);
    tem = fcvt(datab[5], 0, &dec, &sign);
    outtextxy(10,15,tem);
    pres = fcvt(datab[6], 0, &dec, &sign);
    outtextxy(10,35,pres);
}

void r_stdcond(char *filename)
{
    FILE *fp;

    if((fp=fopen(filename,"rb"))==NULL){
        printf("cannot open file\n");
        exit(1);
    }

    fread(datab, sizeof(datab), 1, fp);
    fclose(fp);
}

```

```

// *****
//
//          MODULE: "SITUCHIP.C"
//
//
// *****

# include <stdarg.h>
# include <stdio.h>
# include <dos.h>
# include <bios.h>
# include <string.h>

char *quality_selection(float xx)
{
    char *select;
    if (xx == 1.0)
        select="Good Quality";
    if (xx == 2.0)
        select="Old Chips";
    if (xx == 3.0)
        select="More 10% Hardwood";
    if (xx == 4.0)
        select="Old & More 10% Hardwood";

    return select;
}

char *size_selection(float xx)
{
    char *select;
    if (xx == 1.0)
        select="Standard Size";
    if (xx == 2.0)
        select="Small Chips";
    if (xx == 3.0)
        select="Large Chips";

    return select;
}

```


VITA

NAME: *Mr. JEAN CORBIN*

PLACE OF BIRTH: *EDMUNDSTON, NEW BRUNSWICK*

YEAR OF BIRTH: *1965*

POST-SECONDARY EDUCATION: *B.Sc. BIOCHEMISTRY
B.A.Sc. CHEMICAL ENGINEERING
(UNIV. OF OTTAWA)*

HONOURS AND AWARDS: *DEAN LIST - (UNIV. OF OTTAWA) 1988-89*

RELATED WORK EXPERIENCE: *FRASER INC. (PULP & PAPER)
EDMUNDSTON, NEW BRUNSWICK*

PUBLICATIONS: *M. RAO, Y. YING, J. CORBIN,
"INTELLIGENT ENGINEERING
APPROACH TO PULP AND PAPER FOR
PROCESS CONTROL", CPPA 77TH Annual
Meeting, Montreal, Que., pp. 195-199, 1991*

*J. CORBIN & M. RAO, "AN INTELLIGENT
SIMULATION SYSTEM FOR MONITORING
A BATCH PULPING PROCESS", IACED,
Calgary, Alberta, pp. 221-223, April 8-10,
1991*